

Contents

[Azure Blob storage documentation](#)

[Overview](#)

[What is Azure Blob storage?](#)

[Compare core storage services](#)

[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 \(v8 SDK\)](#)

[Python](#)

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

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

[JavaScript](#)

[JavaScript \(v12 SDK for Node.js\)](#)

[JavaScript \(v10 SDK for Node.js\)](#)

[JavaScript \(v10 SDK for browser\)](#)

[Go](#)

[PHP](#)

[Ruby](#)

Tutorials

Upload and process an image

- 1 - Upload and retrieve image data in the cloud
- 2 - Trigger Azure Functions to process blobs
- 3 - Secure application data
- 4 - Monitor and troubleshoot storage

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

Encrypt and decrypt blobs using Azure Key Vault

Samples

.NET

Java

Python

JavaScript

Azure PowerShell

Azure CLI

Concepts

Storage accounts

Security

 Security recommendations

 Azure Storage encryption at rest

 Customer-managed keys with Azure Key Vault

 Customer-provided keys on the request

 Authorizing data operations

- [Authorize with Azure AD](#)
- [Authorize with Shared Key](#)
- [Delegate access with shared access signatures \(SAS\)](#)
- [Anonymous access to blobs](#)
- [Authorizing management operations](#)
- [Immutable blobs](#)
- [Advanced threat protection](#)
- [Built-in security controls](#)
- [Use Azure Private Endpoints](#)
- [Data redundancy](#)
 - [Data redundancy](#)
 - [Disaster recovery and failover](#)
 - [Design highly available applications](#)
- [Access and performance tiers](#)
 - [Access tiers](#)
 - [Rehydrate data from archive tier](#)
 - [Performance tiers](#)
 - [Manage the Azure Blob storage lifecycle](#)
- [Performance, scaling, and cost optimization](#)
 - [Performance and scalability checklist](#)
 - [Latency in Blob storage](#)
 - [Azure Storage reserved capacity](#)
 - [Plan and manage costs](#)
- [Concurrency](#)
 - [Search and understand blob data](#)
 - [Use AI to understand blob data](#)
 - [Full text search for blob data](#)
- [Data migration](#)
 - [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](#)
- [Import/Export data](#)
 - [Review requirements](#)
 - [Import data to blobs](#)
 - [Import data to files](#)
 - [Export data from blobs](#)
- [How-to](#)
 - [Use customer managed key](#)
 - [View drive status](#)
 - [Review job status](#)
 - [Repair import job](#)
 - [Repair export job](#)
- [FAQ](#)
- [REST API](#)
- [Contact Support](#)
- [Archive](#)
 - [Prepare drives for import \(v1\)](#)
 - [Prepare drives for import \(v2\)](#)
- [Monitor and troubleshoot](#)
 - [Troubleshooting tutorial](#)
 - [Monitor, diagnose, and troubleshoot](#)
 - [Monitor in the Azure portal](#)
 - [Metrics and logging](#)
 - [Metrics in Azure Monitor](#)
 - [Migrate to new metrics](#)
 - [Storage analytics](#)
 - [Storage analytics metrics \(classic\)](#)
 - [Storage analytics logs](#)
 - [Event handling](#)
 - [Change feed](#)
 - [Page blob features](#)
 - [Static websites](#)

How to

[Create and manage storage accounts](#)

[Create a general-purpose storage account](#)

[Create a block blob storage account](#)

[Upgrade a storage account](#)

[Get account type and SKU name \(.NET\)](#)

[Create and manage containers](#)

[Create or delete a container \(.NET\)](#)

[List containers \(.NET\)](#)

[Manage properties and metadata \(.NET\)](#)

[Create and manage blobs](#)

[List blobs \(.NET\)](#)

[Manage blob properties and metadata \(.NET\)](#)

[Copy a blob \(.NET\)](#)

[Create and manage blob snapshots \(.NET\)](#)

[Secure blob data](#)

[Authorize access to blob data](#)

[Authorization options for users](#)

[Portal](#)

[PowerShell](#)

[Azure CLI](#)

[Manage access rights with RBAC](#)

[Portal](#)

[PowerShell](#)

[Azure CLI](#)

[Authenticate and authorize with Azure AD](#)

[Authorize with a managed identity](#)

[Authorize from an application](#)

[Authorize with Shared Key](#)

[View and manage account keys](#)

[Configure connection strings](#)

[Use the Azure Storage REST API](#)

Delegate access with shared access signatures (SAS)

Create a user delegation SAS

Create a service SAS (.NET)

Create an account SAS (.NET)

Define a stored access policy

Manage Azure Storage encryption

Check whether a blob is encrypted

Manage encryption keys for the storage account

Check the encryption key model for the account

Configure customer-managed encryption keys

Provide an encryption key on a request

Configure client-side encryption

.NET

Java

Python

Configure network security

Require secure transfer

Configure firewalls and virtual networks

Enable Transport Layer Security

Set and manage immutability policies

Change how data is replicated

Disaster recovery

Check the Last Sync Time property

Initiate account failover

Recover deleted blobs

Use a storage emulator

Use the Azurite open-source emulator

Use the Azure Storage emulator

Host a static website

Host a static website

Integrate with Azure CDN

Use a custom domain

[Route events to a custom endpoint](#)

[Process change feed logs](#)

[Transfer data](#)

[AzCopy](#)

[Get started](#)

[Use with Blobs](#)

[Use with Amazon S3 buckets](#)

[Configure, optimize, troubleshoot](#)

[Azure Data Factory](#)

[Mount Blob storage as a file system on Linux](#)

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

[Storage migration FAQ](#)

[Develop with blobs](#)

[C++](#)

[iOS](#)

[Java](#)

[Use the Spring Boot Starter](#)

[Xamarin](#)

[Move across regions](#)

[Troubleshoot](#)

[Latency issues](#)

[Reference](#)

[Scalability and performance targets](#)

[Blob storage](#)

[Standard storage accounts](#)

[Premium block blob storage accounts](#)

[Premium page blob storage accounts](#)

[Storage resource provider](#)

[Azure PowerShell](#)

[Azure CLI](#)

[.NET](#)

[Blobs \(version 12.x\)](#)

[Data Movement](#)

[Storage Resource Provider](#)

[Java](#)

[Blobs \(version 12.x\)](#)

[Blobs \(version 8.x\)](#)

[Storage Resource Provider](#)

[JavaScript \(version 12.x\)](#)

[Python \(version 12.x\)](#)

[REST](#)

[Blobs, Queues, Tables, and Files](#)

[Storage Resource Provider](#)

[Import/Export](#)

[C++](#)

[Ruby](#)

[PHP](#)

[iOS](#)

[Android](#)

[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](#)

[Resource Manager template](#)

Resources

[Azure updates](#)
[Azure Storage Explorer](#)
 [Download Storage Explorer](#)
 [Get started with Storage Explorer](#)
 [Storage Explorer release notes](#)
 [Troubleshoot Storage Explorer](#)
 [Storage Explorer accessibility](#)

[Azure Storage forum](#)
[Azure Storage on Stack Overflow](#)
[Pricing for Blob storage](#)
[Azure pricing calculator](#)

Videos

[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](#)

[Quickstarts](#)

[Analyze data with Databricks](#)

[Tutorials](#)

[Extract, transform, and load data using Azure Databricks](#)

[Access Data Lake Storage data with Databricks using Spark](#)

[Extract, transform, load data with Apache Hive on Azure HDInsight](#)

[Insert data into Databricks Delta tables by using Event Grid](#)

[Samples](#)

[.NET](#)

[Java](#)

[Python](#)

[JavaScript](#)

[Concepts](#)

[Processing big data](#)

[Integrating with Azure services](#)

[Filtering data with query acceleration](#)

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

[Azure Blob File System URI](#)

[About hierarchical namespaces](#)

[About storage accounts](#)

[Best practices](#)

[Data redundancy](#)

[Data transfer solutions](#)

[Concurrency](#)

[Security](#)

[Authorizing data operations](#)

[Authorizing management operations](#)

[Access control \(directories and files\)](#)

[Azure Storage encryption at rest](#)

[Customer-managed keys with Azure Key Vault](#)

[Customer-provided keys on the request](#)

[Advanced threat protection](#)

[Use Azure Private Endpoints](#)

[Compatibility](#)

[Multi-protocol access](#)

[Supported Blob storage features](#)

[Supported Azure services](#)

[Supported open source platforms](#)

[Monitoring and troubleshooting](#)

[Monitor, diagnose, and troubleshoot](#)

[Metrics in Azure Monitor](#)

[Migrate to new metrics](#)

[Storage analytics logs](#)

[How to](#)

[Create](#)

[Create a general purpose v2 storage account](#)

[Migrate](#)

[Migrate from Data Lake Storage Gen1](#)

[Migrate an on-premises HDFS store](#)

[Secure](#)

Authorize access to data

Authorization options for users

Portal

PowerShell

Azure CLI

Manage access rights with RBAC

Portal

PowerShell

Azure CLI

Authenticate and authorize with Azure AD

Authorize with a managed identity

Authorize from an application

Authorize with Shared Key

View and manage account keys

Configure connection strings

Use the Azure Storage REST API

Manage Azure Storage encryption

Manage encryption keys for the storage account

Check the encryption key model for the account

Configure customer-managed encryption keys

Configure client-side encryption

.NET

Java

Python

Configure network security

Require secure transfer

Configure firewalls and virtual networks

Enable Transport Layer Security

Manage

.NET

Java

Python

- [JavaScript](#)
- [PowerShell](#)
- [CLI](#)
- [Storage Explorer](#)
- [Hadoop File System CLI](#)
- [Query](#)
 - [Filter data with query acceleration](#)
- [Transfer](#)
 - [AzCopy](#)
 - [Get started](#)
 - [Use with Data Lake Storage](#)
 - [Configure, optimize, troubleshoot](#)
 - [Azure Data Factory](#)
 - [Transfer data with the Data Movement library](#)
- [Optimize](#)
 - [Optimize performance](#)
 - [Optimize Spark on HDInsight](#)
 - [Optimize Hive on HDInsight](#)
 - [Optimize MapReduce on HDInsight](#)
 - [Optimize Storm on HDInsight](#)
- [Recover](#)
 - [Check the Last Sync Time property](#)
- [Emulate](#)
 - [Use the Azurite open-source emulator](#)
 - [Use the Azure Storage emulator](#)
- [Reference](#)
 - [.NET](#)
 - [Java](#)
 - [Python](#)
 - [JavaScript](#)
 - [REST](#)
 - [Query acceleration](#)

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](#)

Resource Manager template

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 accessibility](#)

Forum

[Azure Storage on Stack Overflow](#)

Pricing

[Azure pricing calculator](#)

[Pricing calculator](#)

[Service updates](#)

[Stack Overflow](#)

Videos

What is Azure Blob storage?

11/5/2019 • 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 the core Azure Storage services

4/16/2020 • 11 minutes to read • [Edit Online](#)

The Azure Storage platform is Microsoft's cloud storage solution for modern data storage scenarios. Core storage services offer a massively scalable object store for data objects, disk storage for Azure virtual machines (VMs), a file system service for the cloud, a messaging store for reliable messaging, and a NoSQL store. The services are:

- **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 datacenters 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.

Core storage 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](#).

Example scenarios

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

FEATURE	DESCRIPTION	WHEN TO USE
---------	-------------	-------------

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.
- Diagnostic 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 and queue data.** Azure Storage supports authentication and authorization with Azure AD for the Blob and Queue services via role-based access control (RBAC). Authorizing requests with Azure AD is recommended for superior security and ease of use. For more information, see [Authorize access to Azure blobs and queues using Azure Active Directory](#).
- **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 the core storage services. 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, the core Azure Storage services 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 Storage Client Tools](#)
- [Azure Developer Tools](#)

Next steps

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

Introduction to Azure Blob storage

4/3/2020 • 4 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

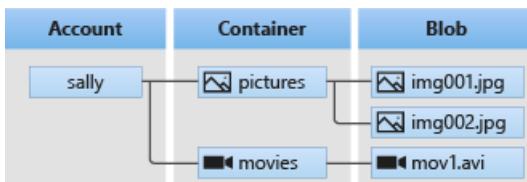
For more information about Data Lake Storage Gen2, see [Introduction to Azure Data Lake Storage Gen2](#).

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 Azure Storage blob endpoint forms the base address for the objects in your storage account.

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

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

To create a storage account, see [Create a storage account](#). To learn more about storage accounts, see [Azure storage account overview](#).

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.

NOTE

The container name must be lowercase. 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, up to about 4.7 TB. Block blobs are made up of blocks of data that can be managed individually.
- **Append blobs** are made up of blocks like block blobs, but are optimized for append operations. Append blobs are ideal for scenarios such as logging data from virtual machines.
- **Page blobs** store random access files up to 8 TB in size. Page blobs store virtual hard drive (VHD) files and serve as disks for Azure virtual machines. 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](#).

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 \(Preview\)](#).
- 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

4/17/2020 • 2 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

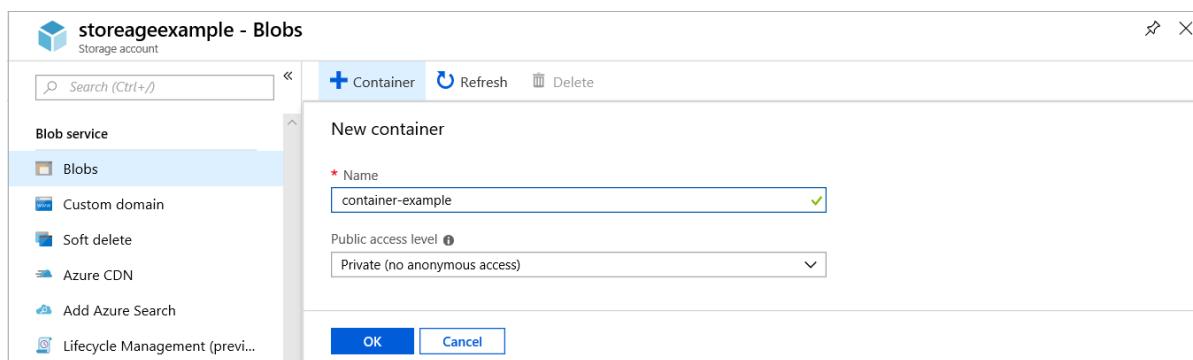
To access Azure Storage, you'll need an Azure subscription. If you don't already have a subscription, create a [free account](#) before you begin.

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 **Blob service** 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 **OK** to create the container.



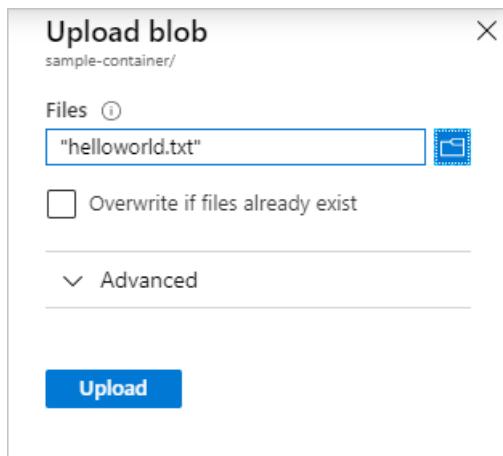
Upload a block blob

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.



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**.

Clean up resources

To remove the resources you created in this quickstart, you can 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 transfer files between a local disk and Azure Blob storage with Azure portal. To learn more about working with Blob storage, continue to the Blob storage How-to.

[Blob Storage Operations How-To](#)

Quickstart: Use Azure Storage Explorer to create a blob

3/20/2020 • 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](#).

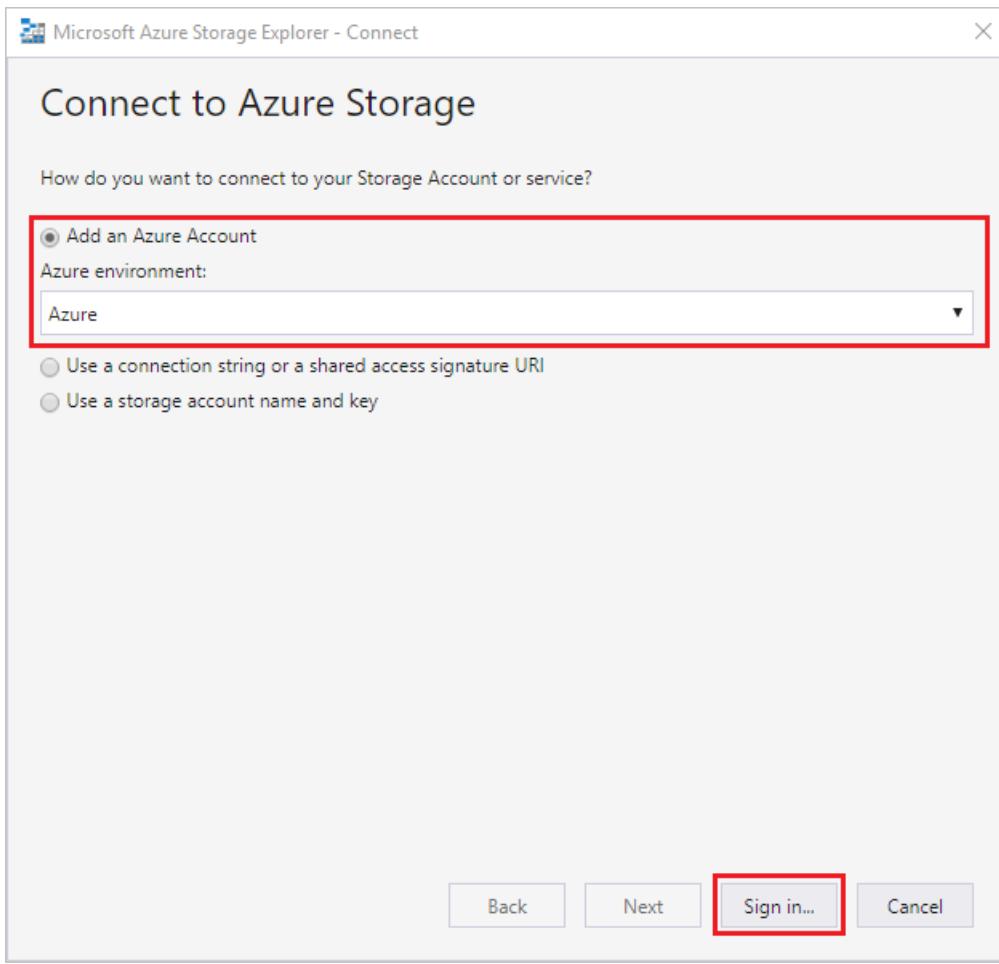
This quickstart requires that you install Azure Storage Explorer. To install Azure Storage Explorer for Windows, Macintosh, or Linux, see [Azure Storage Explorer](#).

Log in to Storage Explorer

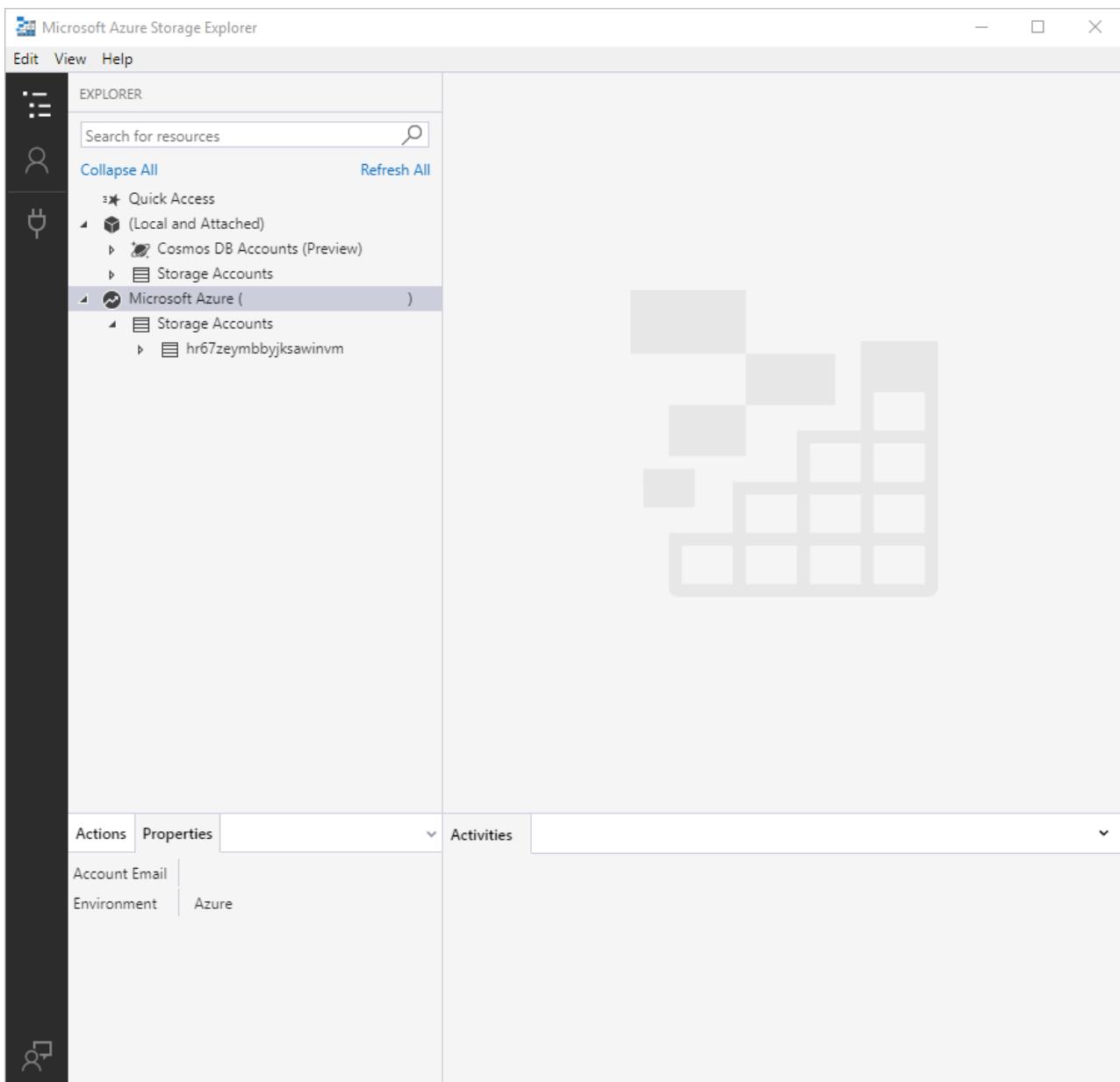
On first launch, the **Microsoft Azure Storage Explorer - Connect** window is shown. Storage Explorer provides several ways to connect to storage accounts. The following table lists the different ways you can connect:

TASK	PURPOSE
Add an Azure Account	Redirects you to your organization's sign-in page to authenticate you to Azure.
Use a connection string or shared access signature URI	Can be used to directly access a container or storage account with a SAS token or a shared connection string.
Use a storage account name and key	Use the storage account name and key of your storage account to connect to Azure storage.

Select **Add an Azure Account** and click **Sign in...**. Follow the on-screen prompts to sign into your Azure account.



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 [Azure Storage Emulator](#), [Cosmos DB](#) accounts, or [Azure Stack](#) environments.



Create a container

Blobs are always uploaded into a container. This allows you to organize groups of blobs like you organize your files on your computer in folders.

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

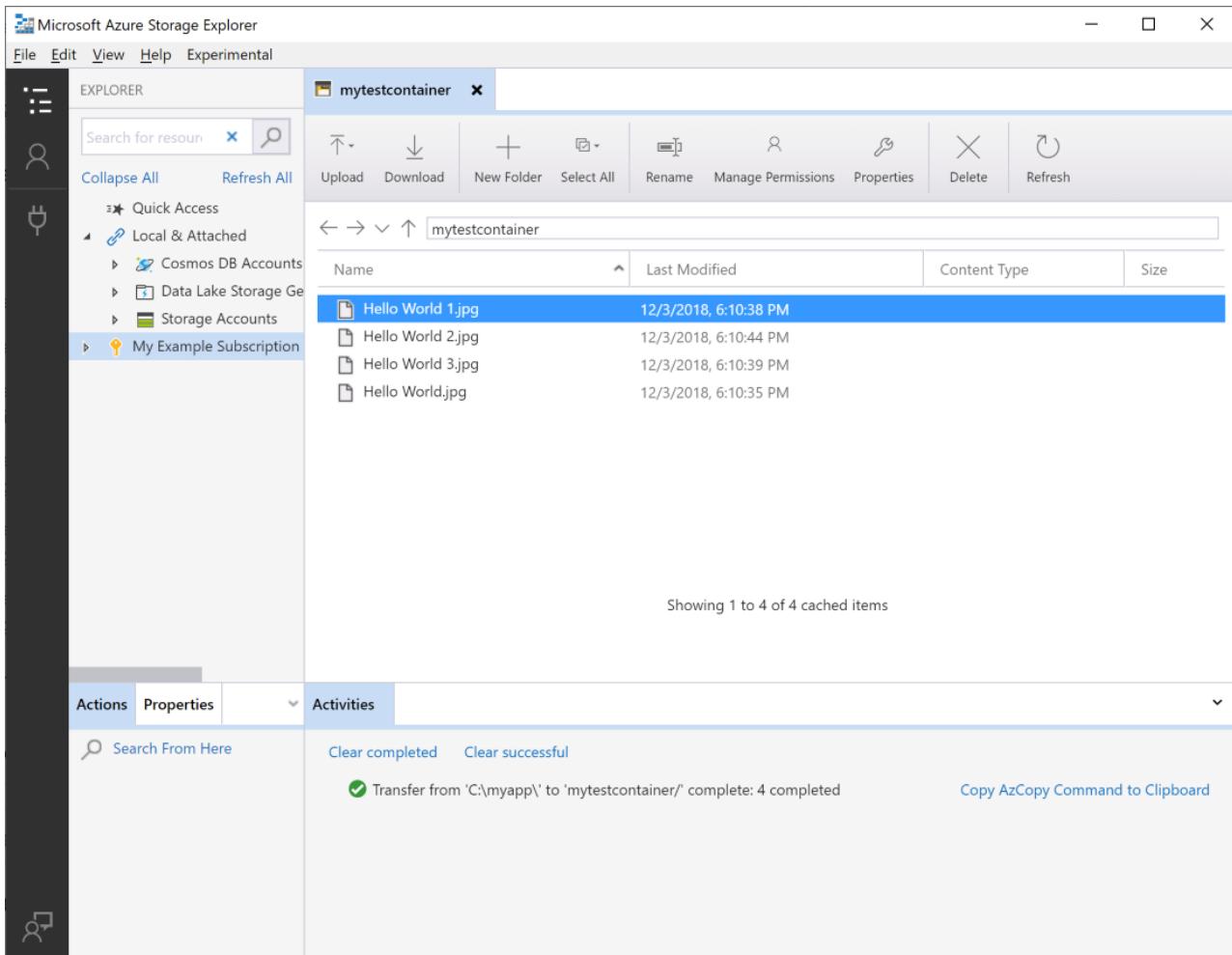
Blob storage supports block blobs, append blobs, and page blobs. VHD files used to back IaaS VMs are page blobs. Append blobs are used for logging, such as when you want to write to a file and then keep adding more information. Most files stored in Blob storage are block blobs.

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 .vhd or .vhdx file, choose **Upload .vhd/.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 OK, 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.

The screenshot shows the Microsoft Azure Storage Explorer interface. The left sidebar has sections for 'EXPLORER', 'Quick Access', '(Local and Attached)', 'Storage Accounts', and 'Microsoft Azure'. Under 'Microsoft Azure', there's a 'Storage Accounts' section with a 'hr67zeymbbyjksawinvvm' account expanded, showing 'Blob Containers', 'File Shares', 'Queues', and 'Tables'. The 'mytestcontainer' blob container is selected. The main pane displays a table of blobs with columns: Name, Last Modified, Blob Type, Content Type, and Size. The table shows four entries: 'HelloWorld.jpg' (Mon, 20 Nov 2017 18:43:27 GMT), 'HelloWorld1.jpg' (Mon, 20 Nov 2017 21:01:32 GMT), 'HelloWorld2.jpg' (Mon, 20 Nov 2017 21:21:29 GMT), and 'HelloWorld3.jpg' (Mon, 20 Nov 2017 21:21:39 GMT). All are Block Blob type, image/jpeg content type, and 3.5 MB size. A red box highlights this table. Below it, a message says 'Showing 1 to 4 of 4 cached items'. The 'Actions' pane shows URL, Type (Blob Container), Public Read Access (Off), Lease State (available), Lease Status (unlocked), and Last Modified (Mon, 20 Nov 2017 18:11:49 GM). The 'Properties' pane shows the URL: <https://hr67zeymbbyjksawinvvm.blob.core.windows.net/mytestcontainer>. The 'Activities' pane lists three completed upload activities.

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.

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 Snapshots**. A list of the snapshots for the blob are shown in the current tab.

The screenshot shows the 'mytestcontainer' blob container in the Azure Storage Explorer. The 'Actions' bar includes 'Download', 'Open', 'Copy URL', 'Select All', 'Copy', 'Delete', 'Make Snapshot', 'Promote Snapshot', 'Properties', and 'More'. The main pane shows a table of snapshots for 'HelloWorld.jpg'. The table has columns: Name, Last Modified, Blob Type, Content Type, Size, Lease State, Disk Name, and VM Name. The table shows three entries: 'HelloWorld.jpg (2017-11-21T16:57:09.2640664Z)' (Mon, 20 Nov 2017 18:43:27 GMT), 'HelloWorld.jpg (2017-11-21T18:17:5814386Z)' (Mon, 20 Nov 2017 18:43:27 GMT), and 'HelloWorld.jpg' (Mon, 20 Nov 2017 18:43:27 GMT). All are Block Blob type, image/jpeg content type, and 3.5 MB size. The first snapshot is highlighted with a red box.

Manage access policies

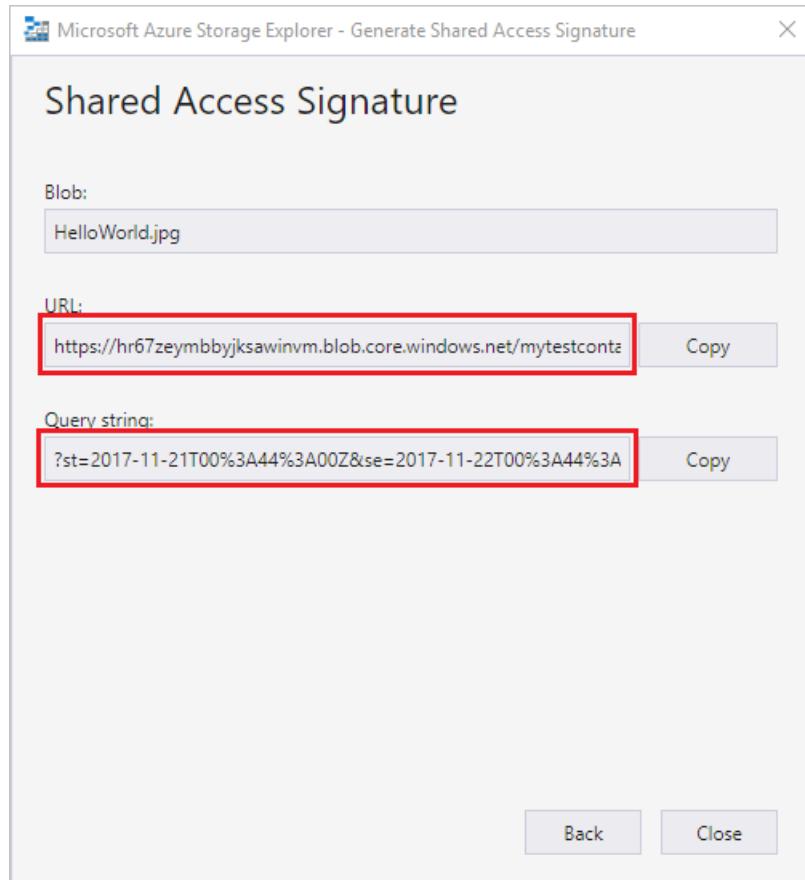
Storage Explorer provides the ability to manage access policies for containers within its user interface. There are two types of secure access policies (SAS), service level and account level. Account level SAS targets the storage account and can apply to multiple services and resources. Service level SAS are defined on a resource under a particular service. To generate a service level SAS, right-click any container and select **Manage Access Policies....**. To generate an account level SAS, right-click on the storage account.

Select **Add** to add a new access policy and define the permissions for the policy. When complete select **Save** to

save the access policy. This policy is now available for use when configuring a Shared Access Signature.

Work with Shared Access Signatures

Shared Access Signatures (SAS) can be retrieved through Storage Explorer. 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**. The full URL with the query string as well as the query string by itself are provided and can be copied from the next screen.



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 How-to.

[Blob Storage Operations How-To](#)

Quickstart: Upload, download, and list blobs with PowerShell

3/31/2020 • 4 minutes to read • [Edit Online](#)

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

NOTE

The features described in this article are now available to accounts that have a hierarchical namespace. To review limitations, see the [Known issues with Azure Data Lake Storage Gen2](#) article.

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.

NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

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 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).

```
$storageAccount = New-AzStorageAccount -ResourceGroupName $resourceGroup ` 
    -Name "mystorageaccount" ` 
    -SkuName Standard_LRS ` 
    -Location $location ` 

$ctx = $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 $ctx -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:_Test\Images* folder on the local disk to the container you created.

```
# upload a file
Set-AzStorageBlobContent -File "D:\_TestImages\Image001.jpg" ` 
    -Container $containerName ` 
    -Blob "Image001.jpg" ` 
    -Context $ctx

# upload another file
Set-AzStorageBlobContent -File "D:\_TestImages\Image002.png" ` 
    -Container $containerName ` 
    -Blob "Image002.png" ` 
    -Context $ctx
```

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 $ctx | select 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:_TestImages\Downloads* on the local disk.

```
# download first blob
Get-AzStorageBlobContent -Blob "Image001.jpg" ` 
    -Container $containerName ` 
    -Destination "D:\_TestImages\Downloads\" ` 
    -Context $ctx

# download another blob
Get-AzStorageBlobContent -Blob "Image002.png" ` 
    -Container $containerName ` 
    -Destination "D:\_TestImages\Downloads\" ` 
    -Context $ctx
```

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

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 disk and Azure Blob storage. To learn more about working

with Blob storage by using PowerShell, continue to How-to use Azure PowerShell with Azure Storage.

Using Azure PowerShell with Azure 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

3/20/2020 • 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.

NOTE

The features described in this article are now available to accounts that have a hierarchical namespace. To review limitations, see the [Known issues with Azure Data Lake Storage Gen2](#) article.

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](#).

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 https://shell.azure.com , 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.

Install the Azure CLI locally

If you choose to install and use the Azure CLI locally, this quickstart requires that you are running the Azure CLI version 2.0.46 or later. Run `az --version` to determine your version. If you need to install or upgrade, see [Install the Azure CLI](#).

If you are running the Azure CLI locally, you must log in and authenticate. This step is not necessary if you are using Azure Cloud Shell. To log in to Azure CLI, run `az login` and authenticate in the browser window:

```
az login
```

For more information about authentication` with Azure CLI, see [Sign in with Azure CLI](#).

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.

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

```
az storage container create \
--account-name <storage-account> \
--name <container> \
--auth-mode login
```

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 blobs in Azure Storage, continue to the tutorial for working with Azure Blob storage.

[How to: Blob storage operations with the Azure CLI](#)

Quickstart: Azure Blob storage client library v12 for .NET

3/11/2020 • 8 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.

NOTE

To get started with the previous SDK version, see [Quickstart: Azure Blob storage client library for .NET](#).

Use the Azure Blob storage client library v12 for .NET 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

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

NOTE

The features described in this article are now available to accounts that have a hierarchical namespace. To review limitations, see the [Known issues with Azure Data Lake Storage Gen2](#) article.

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
```

- In side 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:

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

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. View your storage account credentials by following these steps:

- Sign in to the [Azure portal](#).
- Locate your storage account.
- 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.
- 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.

Storage account name	<code>storagesamples</code>	
key1		
Key	<code><account-key></code>	
Connection string	<code><connection-string></code>	

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>"
```

Restart programs

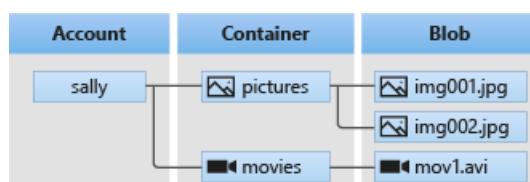
After you add the environment variable, restart any running programs that will need to read the environment variable. For example, restart your development environment or editor before continuing.

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 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.

- **BlobDownloadInfo:** The `BlobDownloadInfo` class represents the properties and content returned from downloading a blob.

Code examples

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

- [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:

```
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 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 [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);

// Open the file and upload its data
using FileStream uploadFileStream = File.OpenRead(localFilePath);
await blobClient.UploadAsync(uploadFileStream, true);
uploadFileStream.Close();
```

List the 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 blobs

Download the previously created blob by calling the [DownloadAsync](#) 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 "DOWNLOAD" before the .txt extension
// so you can compare the files in the data directory
string downloadFilePath = localFilePath.Replace(".txt", "DOWNLOAD.txt");

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

// Download the blob's contents and save it to a file
BlobDownloadInfo download = await blobClient.DownloadAsync();

using (FileStream downloadFileStream = File.OpenWrite(downloadFilePath))
{
    await download.Content.CopyToAsync(downloadFileStream);
    downloadFileStream.Close();
}

```

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.Write("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

12/18/2019 • 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.

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

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

NOTE

The features described in this article are now available to accounts that have a hierarchical namespace. To review limitations, see the [Known issues with Azure Data Lake Storage Gen2](#) article.

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 void Main()
        {
            Console.WriteLine("Azure Blob Storage - .NET quickstart sample\n");

            // Run the examples asynchronously, wait for the results before proceeding
            ProcessAsync().GetAwaiter().GetResult();

            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>
```

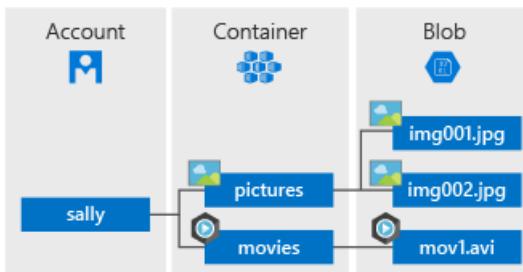
After you add the environment variable, restart any running programs that will need to read the environment variable. For example, restart your development environment or editor before continuing.

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

4/21/2020 • 9 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.

[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](#).

NOTE

To get started with the previous SDK version, see [Quickstart: Manage blobs with Java v8 SDK](#).

NOTE

The features described in this article are now available to accounts that have a hierarchical namespace. To review limitations, see the [Known issues with Azure Data Lake Storage Gen2](#) article.

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.

```
mvn archetype:generate -DgroupId=com.blobs.quickstart \
-DartifactId=blob-quickstart-v12 \
-DarchetypeArtifactId=maven-archetype-quickstart \
-DarchetypeVersion=1.4 \
-DinteractiveMode=false
```

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] -----
[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. Inside 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>12.0.0</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;

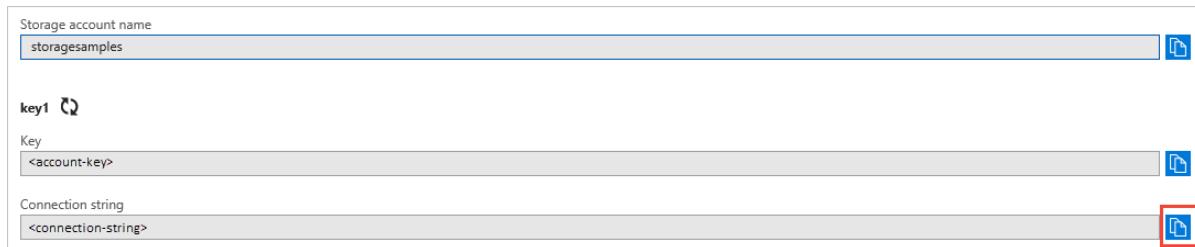
/**
 * Azure blob storage v12 SDK quickstart
 */
import com.azure.storage.blob.*;
import com.azure.storage.blob.models.*;
import java.io.*;

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. 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, 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>"
```

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

### Restart programs

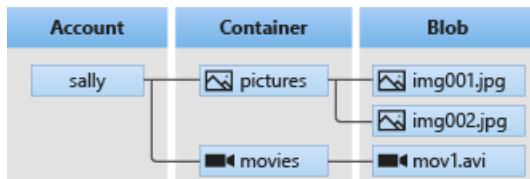
After you add the environment variable, restart any running programs that will need to read the environment variable. For example, restart your development environment or editor before continuing.

## 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 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 `listBlobsFlat`.

## 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

3/31/2020 • 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.

## 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 URIs 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 [CloudBlob.Container.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

3/20/2020 • 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.

[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, 3.5, or above.

### NOTE

To get started with the previous SDK version, see [Quickstart: Manage blobs with Python v2.1 SDK](#).

### NOTE

The features described in this article are now available to accounts that have a hierarchical namespace. To review limitations, see the [Known issues with Azure Data Lake Storage Gen2](#) article.

## 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
```

3. Inside 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

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

try:
 print("Azure Blob storage v12 - 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. 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, 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=""
```

#### macOS

```
export AZURE_STORAGE_CONNECTION_STRING=""
```

#### Restart programs

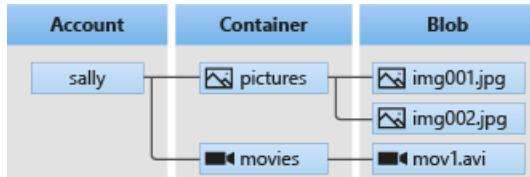
After you add the environment variable, restart any running programs that will need to read the environment variable. For example, restart your development environment or editor before continuing.

## 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 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 perform the following 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 connection string for the storage account 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 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 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 = "quickstart" + 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 text file in the local directory.
2. Gets a reference to a [BlobClient](#) object by calling the `get_blob_client` method on the [BlobServiceClient](#) from the [Create a container](#) section.
3. 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 file in local data directory to upload and download
local_path = "./data"
local_file_name = "quickstart" + 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'))
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().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. 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 `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)

print("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 *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 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 Python.

To see Blob storage sample apps, continue to:

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

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

# Quickstart: Manage blobs with Python v2.1 SDK

1/29/2020 • 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.

## 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](#).

### NOTE

The features described in this article are now available to accounts that have a hierarchical namespace. To review limitations, see the [Known issues with Azure Data Lake Storage Gen2](#) article.

## 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

3/5/2020 • 8 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. You'll upload, download, and list blobs, and you'll create and delete containers.

[API reference documentation](#) | [Library source code](#) | [Package \(Node Package Manager\)](#) | [Samples](#)

## Prerequisites

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

### NOTE

To get started with the previous SDK version, see [Quickstart: Manage blobs with JavaScript v10 SDK in Node.js](#).

### NOTE

The features described in this article are now available to accounts that have a hierarchical namespace. To review limitations, see the [Known issues with Azure Data Lake Storage Gen2](#) article.

## Setting up

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

### Create the 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 new text file called *package.json*. This file defines the Node.js project. Save this file in the *blob-quickstart-v12* directory. Here is the contents of the file:

```
{
 "name": "blob-quickstart-v12",
 "version": "1.0.0",
 "description": "Use the @azure/storage-blob SDK version 12 to interact with Azure Blob storage",
 "main": "blob-quickstart-v12.js",
 "scripts": {
 "start": "node blob-quickstart-v12.js"
 },
 "author": "Your Name",
 "license": "MIT",
 "dependencies": {
 "@azure/storage-blob": "^12.0.0",
 "@types/dotenv": "^4.0.3",
 "dotenv": "^6.0.0"
 }
}
```

You can put your own name in for the `author` field, if you'd like.

## Install the package

While still in the `blob-quickstart-v12` directory, install the Azure Blob storage client library for JavaScript package by using the `npm install` command. This command reads the `package.json` file and installs the Azure Blob storage client library v12 for JavaScript package and all the libraries on which it depends.

```
npm install
```

## Set up the app framework

From the project directory:

1. Open another new text file in your code editor
2. Add `require` calls to load Azure and Node.js modules
3. Create the structure for the program, including basic exception handling

Here's the code:

```
const { BlobServiceClient } = require('@azure/storage-blob');
const uuidv1 = require('uuid/v1');

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));
```

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

## Copy your credentials from the Azure portal

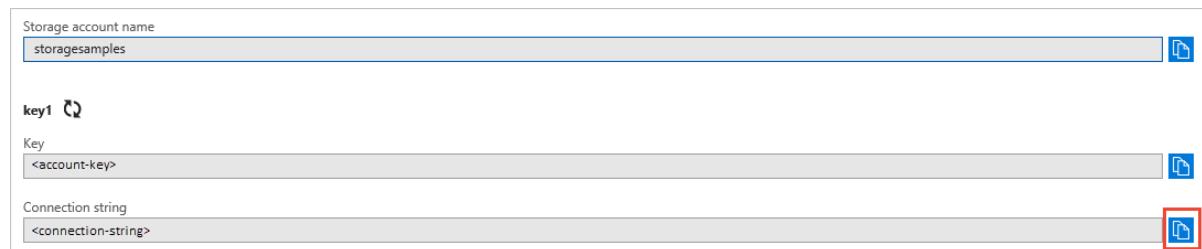
When the sample application makes a request to Azure Storage, it must be authorized. To authorize a request, add your storage account credentials to the application as a connection string. View your storage account credentials by following these steps:

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, 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>"
```

### Restart programs

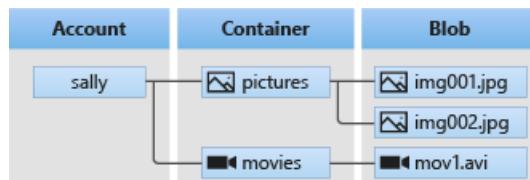
After you add the environment variable, restart any running programs that will need to read the environment variable. For example, restart your development environment or editor before continuing.

## 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 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.

## Code examples

These example code snippets show you how to perform the following with the Azure Blob storage client 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](#)

### 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:

```
// 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.
const AZURE_STORAGE_CONNECTION_STRING = process.env.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](#).

Create 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.

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 = await 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 = await blobServiceClient.getContainerClient(containerName);

// Create the container
const createContainerResponse = await containerClient.create();
console.log("Container was created successfully. requestId: ", createContainerResponse.requestId);
```

## Upload blobs to a container

The following code snippet:

1. Creates a text string to upload to a blob.
2. Gets a reference to a [BlockBlobClient](#) object by calling the [getBlockBlobClient](#) method on the [ContainerClient](#) from the [Create a container](#) section.
3. Uploads the text string data to the blob by calling the [upload](#) method.

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

```
// 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);
```

## List the blobs in a container

List the blobs in the container by calling the [listBlobsFlat](#) 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` function:

```
console.log('\nListing blobs...');

// List the blob(s) in the container.
for await (const blob of containerClient.listBlobsFlat()) {
 console.log('\t', blob.name);
}
```

## Download blobs

Download the previously created blob by calling the [download](#) method. The example code includes a helper function called `streamToString`, which is used to read a Node.js readable stream into a string.

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

```
// 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));
```

Add this helper function *after* the `main` function:

```
// A helper function used to read a Node.js readable stream into a string
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);
 });
}
```

## Delete a container

The following 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.

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

```
console.log('\nDeleting container...');

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

## Run the code

This app creates a text string and uploads it to Blob storage. The example then lists the blob(s) in the container, downloads the blob, and displays the downloaded data.

From a console prompt, navigate to the directory containing the `blob-quickstart-v12.py` file, then execute the following `node` command to run the app.

```
node blob-quickstart-v12.js
```

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.

## 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 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: Manage blobs with JavaScript v10 SDK in Node.js

3/3/2020 • 8 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. You'll upload, download, list, and delete blobs, and you'll manage containers.

## Prerequisites

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

## Download the sample application

The [sample application](#) in this quickstart is a simple Node.js console application. To begin, clone the repository to your machine using the following command:

```
git clone https://github.com/Azure-Samples/azure-storage-js-v10-quickstart.git
```

Next, change folders for the application:

```
cd azure-storage-js-v10-quickstart
```

Now, open the folder in your favorite code editing environment.

## Configure your storage credentials

Before running the application, you must provide the security credentials for your storage account. The sample repository includes a file named `.env.example`. Rename this file by removing the `.example` extension, which results in a file named `.env`. Inside the `.env` file, add your account name and access key values after the `AZURE_STORAGE_ACCOUNT_NAME` and `AZURE_STORAGE_ACCOUNT_ACCESS_KEY` keys.

## Install required packages

In the application directory, run `npm install` to install the required packages for the application.

```
npm install
```

## Run the sample

Now that the dependencies are installed, you can run the sample by issuing the following command:

```
npm start
```

The output from the app will be similar to the following example:

```
Container "demo" is created
Containers:
- container-one
- container-two
- demo
Blob "quickstart.txt" is uploaded
Local file "./readme.md" is uploaded
Blobs in "demo" container:
- quickstart.txt
- readme-stream.md
- readme.md
Blob downloaded blob content: "hello!"
Blob "quickstart.txt" is deleted
Container "demo" is deleted
Done
```

If you're using a new storage account for this quickstart, then you may only see the *demo* container listed under the label "*Containers*".

## Understanding the code

The sample begins by importing a number of classes and functions from the Azure Blob storage namespace. Each of the imported items is discussed in context as they're used in the sample.

```
const {
 Aborter,
 BlobURL,
 BlockBlobURL,
 ContainerURL,
 ServiceURL,
 SharedKeyCredential,
 StorageURL,
 uploadStreamToBlockBlob
} = require('@azure/storage-blob');
```

Credentials are read from environment variables based on the appropriate context.

```
if (process.env.NODE_ENV !== 'production') {
 require('dotenv').config();
}
```

The *dotenv* module loads environment variables when running the app locally for debugging. Values are defined in a file named *.env* and loaded into the current execution context. In production, the server configuration provides these values, which is why this code only runs when the script is *not* running under a "production" environment.

The next block of modules is imported to help interface with the file system.

```
const fs = require('fs');
const path = require('path');
```

The purpose of these modules is as follows:

- *fs* is the native Node.js module used to work with the file system
- *path* is required to determine the absolute path of the file, which is used when uploading a file to Blob storage

Next, environment variable values are read and set aside in constants.

```
const STORAGE_ACCOUNT_NAME = process.env.AZURE_STORAGE_ACCOUNT_NAME;
const ACCOUNT_ACCESS_KEY = process.env.AZURE_STORAGE_ACCOUNT_ACCESS_KEY;
```

The next set of constants helps to reveal the intent of file size calculations during upload operations.

```
const ONE_MEGABYTE = 1024 * 1024;
const FOUR_MEGABYTES = 4 * ONE_MEGABYTE;
```

Requests made by the API can be set to time out after a given interval. The [Aborter](#) class is responsible for managing how requests are timed-out and the following constant is used to define timeouts used in this sample.

```
const ONE_MINUTE = 60 * 1000;
```

## Calling code

To support JavaScript's *async/await* syntax, all the calling code is wrapped in a function named *execute*. Then *execute* is called and handled as a promise.

```
async function execute() {
 // commands...
}

execute().then(() => console.log("Done")).catch((e) => console.log(e));
```

All of the following code runs inside the *execute* function where the `// commands...` comment is placed.

First, the relevant variables are declared to assign names, sample content and to point to the local file to upload to Blob storage.

```
const containerName = "demo";
const blobName = "quickstart.txt";
const content = "hello!";
const localFilePath = "./readme.md";
```

Account credentials are used to create a pipeline, which is responsible for managing how requests are sent to the REST API. Pipelines are thread-safe and specify logic for retry policies, logging, HTTP response deserialization rules, and more.

```
const credentials = new SharedKeyCredential(STORAGE_ACCOUNT_NAME, ACCOUNT_ACCESS_KEY);
const pipeline = StorageURL.newPipeline(credentials);
const serviceURL = new ServiceURL(`https://${STORAGE_ACCOUNT_NAME}.blob.core.windows.net`, pipeline);
```

The following classes are used in this block of code:

- The [SharedKeyCredential](#) class is responsible for wrapping storage account credentials to provide them to a request pipeline.
- The [StorageURL](#) class is responsible for creating a new pipeline.
- The [ServiceURL](#) models a URL used in the REST API. Instances of this class allow you to perform actions like list containers and provide context information to generate container URLs.

The instance of *ServiceURL* is used with the [ContainerURL](#) and [BlockBlobURL](#) instances to manage containers and

blobs in your storage account.

```
const containerURL = ContainerURL.fromServiceURL(serviceURL, containerName);
const blockBlobURL = BlockBlobURL.fromContainerURL(containerURL, blobName);
```

The *containerURL* and *blockBlobURL* variables are reused throughout the sample to act on the storage account.

At this point, the container doesn't exist in the storage account. The instance of *ContainerURL* represents a URL that you can act upon. By using this instance, you can create and delete the container. The location of this container equates to a location such as this:

```
https://<ACCOUNT_NAME>.blob.core.windows.net/demo
```

The *blockBlobURL* is used to manage individual blobs, allowing you to upload, download, and delete blob content.

The URL represented here is similar to this location:

```
https://<ACCOUNT_NAME>.blob.core.windows.net/demo/quickstart.txt
```

As with the container, the block blob doesn't exist yet. The *blockBlobURL* variable is used later to create the blob by uploading content.

## Using the *Aborter* class

Requests made by the API can be set to time out after a given interval. The *Aborter* class is responsible for managing how requests are timed out. The following code creates a context where a set of requests is given 30 minutes to execute.

```
const aborter = Aborter.timeout(30 * ONE_MINUTE);
```

Aborters give you control over requests by allowing you to:

- designate the amount of time given for a batch of requests
- designate how long an individual request has to execute in the batch
- allow you to cancel requests
- use the *Aborter.none* static member to stop your requests from timing out all together

## Create a container

To create a container, the *ContainerURL*'s *create* method is used.

```
await containerURL.create(aborter);
console.log(`Container: "${containerName}" is created`);
```

As the name of the container is defined when calling *ContainerURL.fromServiceURL(serviceURL, containerName)*, calling the *create* method is all that's required to create the container.

## Show container names

Accounts can store a vast number of containers. The following code demonstrates how to list containers in a segmented fashion, which allows you to cycle through a large number of containers. The *showContainerNames* function is passed instances of *ServiceURL* and *Aborter*.

```
console.log("Containers:");
await showContainerNames(serviceURL, aborter);
```

The `showContainerNames` function uses the `listContainersSegment` method to request batches of container names from the storage account.

```
async function showContainerNames(aborter, serviceURL) {
 let marker = undefined;

 do {
 const listContainersResponse = await serviceURL.listContainersSegment(aborter, marker);
 marker = listContainersResponse.nextMarker;
 for(let container of listContainersResponse.containerItems) {
 console.log(` - ${container.name}`);
 }
 } while (marker);
}
```

When the response is returned, then the `containerItems` are iterated to log the name to the console.

## Upload text

To upload text to the blob, use the `upload` method.

```
await blockBlobURL.upload(aborter, content, content.length);
console.log(`Blob "${blobName}" is uploaded`);
```

Here the text and its length are passed into the method.

## Upload a local file

To upload a local file to the container, you need a container URL and the path to the file.

```
await uploadLocalFile(aborter, containerURL, localFilePath);
console.log(`Local file "${localFilePath}" is uploaded`);
```

The `uploadLocalFile` function calls the `uploadFileToBlockBlob` function, which takes the file path and an instance of the destination of the block blob as arguments.

```
async function uploadLocalFile(aborter, containerURL, filePath) {

 filePath = path.resolve(filePath);

 const fileName = path.basename(filePath);
 const blockBlobURL = BlockBlobURL.fromContainerURL(containerURL, fileName);

 return await uploadFileToBlockBlob(aborter, filePath, blockBlobURL);
}
```

## Upload a stream

Uploading streams is also supported. This sample opens a local file as a stream to pass to the upload method.

```
await uploadStream(containerURL, localFilePath, aborter);
console.log(`Local file "${localFilePath}" is uploaded as a stream`);
```

The `uploadStream` function calls `uploadStreamToBlockBlob` to upload the stream to the storage container.

```

async function uploadStream(aborter, containerURL, filePath) {
 filePath = path.resolve(filePath);

 const fileName = path.basename(filePath).replace('.md', '-stream.md');
 const blockBlobURL = BlockBlobURL.fromContainerURL(containerURL, fileName);

 const stream = fs.createReadStream(filePath, {
 highWaterMark: FOUR_MEGABYTES,
 });

 const uploadOptions = {
 bufferSize: FOUR_MEGABYTES,
 maxBuffers: 5,
 };

 return await uploadStreamToBlockBlob(
 aborter,
 stream,
 blockBlobURL,
 uploadOptions.bufferSize,
 uploadOptions.maxBuffers);
}

```

During an upload, `uploadStreamToBlockBlob` allocates buffers to cache data from the stream in case a retry is necessary. The `maxBuffers` value designates at most how many buffers are used as each buffer creates a separate upload request. Ideally, more buffers equate to higher speeds, but at the cost of higher memory usage. The upload speed plateaus when the number of buffers is high enough that the bottleneck transitions to the network or disk instead of the client.

## Show blob names

Just as accounts can contain many containers, each container can potentially contain a vast amount of blobs. Access to each blob in a container are available via an instance of the `ContainerURL` class.

```

console.log(`Blobs in "${containerName}" container:`);
await showBlobNames(aborter, containerURL);

```

The function `showBlobNames` calls `listBlobFlatSegment` to request batches of blobs from the container.

```

async function showBlobNames(aborter, containerURL) {
 let marker = undefined;

 do {
 const listBlobsResponse = await containerURL.listBlobFlatSegment(Aborter.none, marker);
 marker = listBlobsResponse.nextMarker;
 for (const blob of listBlobsResponse.segment.blobItems) {
 console.log(` - ${blob.name}`);
 }
 } while (marker);
}

```

## Download a blob

Once a blob is created, you can download the contents by using the `download` method.

```

const downloadResponse = await blockBlobURL.download(aborter, 0);
const downloadedContent = await streamToString(downloadResponse.readableStreamBody);
console.log(`Downloaded blob content: "${downloadedContent}"`);

```

The response is returned as a stream. In this example, the stream is converted to a string by using the following

*streamToString* helper function.

```
// A helper method used to read a Node.js readable stream into a string
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);
 });
}
```

## Delete a blob

The *delete* method from a *BlockBlobURL* instance deletes a blob from the container.

```
await blockBlobURL.delete(aborter)
console.log(`Block blob "${blobName}" is deleted`);
```

## Delete a container

The *delete* method from a *ContainerURL* instance deletes a container from the storage account.

```
await containerURL.delete(aborter);
console.log(`Container "${containerName}" is deleted`);
```

## Clean up resources

All data written to the storage account is automatically deleted at the end of the code sample.

## Next steps

This quickstart demonstrates how to manage blobs and containers in Azure Blob storage using Node.js. To learn more about working with this SDK, refer to the GitHub repository.

[Azure Storage v10 SDK for JavaScript repository](#) [Azure Storage JavaScript API Reference](#)

# Quickstart: Manage blobs with JavaScript v10 SDK in browser

4/2/2020 • 12 minutes to read • [Edit Online](#)

In this quickstart, you learn to manage blobs by using JavaScript code running entirely in the browser. Blobs are objects that can hold large amounts of text or binary data, including images, documents, streaming media, and archive data. You'll use required security measures to ensure protected access to your blob storage account.

## Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- An Azure Storage account. [Create a storage account](#).
- A local web server. This article uses [Node.js](#) to open a basic server.
- [Visual Studio Code](#).
- A VS Code extension for browser debugging, such as [Debugger for Chrome](#) or [Debugger for Microsoft Edge](#).

## Setting up storage account CORS rules

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

Return to the Azure portal and select your storage account. To define a new CORS rule, navigate to the **Settings** section and click on the **CORS** link. Next, click the **Add** button to open the **Add CORS rule** window. For this quickstart, you create an open CORS rule:

The screenshot shows the Azure Storage CORS configuration page. The left sidebar lists various storage account settings like Access keys, Geo-replication, and Shared access signature. The 'CORS' option is selected and highlighted in blue. The main pane displays information about CORS, stating it's an HTTP feature for cross-origin resource sharing. It explains that CORS rules are set individually for Blob, File, Queue, and Table services. A 'Learn more' link is provided. Below this, the 'Blob service' tab is selected, showing a table for defining CORS rules. The table has columns for ALLOWED ORIGIN, ALLOWED METHODS, ALLOWED HEADERS, EXPOSED HEADERS, and MAX AGE. Under ALLOWED METHODS, several checkboxes are checked: DELETE, GET, HEAD, MERGE, POST, OPTIONS, PUT, and PATCH. Under EXPOSED HEADERS, the value '86400' is listed. The MAX AGE column contains the value '0'. A 'Save' and 'Discard' button are at the top right of the main pane.

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.
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 (seconds)	86400	The maximum amount of time the browser caches the preflight OPTIONS request. A value of <code>86400</code> allows the cache to remain for a full day.

#### 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.

Next, you use the Azure cloud shell to create a security token.

## 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.

## Create a shared access signature

The shared access signature (SAS) is used by the code running in the browser to authorize requests to Blob storage. 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 \(SAS\)](#).

You can create a SAS using the Azure CLI through the Azure cloud shell, or with the Azure portal or Azure Storage Explorer. The following table describes the parameters you need to provide values for to generate a SAS with the CLI.

PARAMETER	DESCRIPTION	PLACEHOLDER
<i>expiry</i>	The expiration date of the access token in YYYY-MM-DD format. Enter tomorrow's date for use with this quickstart.	<i>FUTURE_DATE</i>
<i>account-name</i>	The storage account name. Use the name set aside in an earlier step.	<i>YOUR_STORAGE_ACCOUNT_NAME</i>
<i>account-key</i>	The storage account key. Use the key set aside in an earlier step.	<i>YOUR_STORAGE_ACCOUNT_KEY</i>

Use the following CLI command, with actual values for each placeholder, to generate a SAS that you can use in your JavaScript code.

```
az storage account generate-sas \
--permissions racwdl \
--resource-types sco \
--services b \
--expiry FUTURE_DATE \
--account-name YOUR_STORAGE_ACCOUNT_NAME \
--account-key YOUR_STORAGE_ACCOUNT_KEY
```

You may find the series of values after each parameter a bit cryptic. These parameter values are taken from the first letter of their respective permission. The following table explains where the values come from:

PARAMETER	VALUE	DESCRIPTION
<i>permissions</i>	racwdl	This SAS allows <i>read</i> , <i>append</i> , <i>create</i> , <i>write</i> , <i>delete</i> , and <i>list</i> capabilities.
<i>resource-types</i>	sco	The resources affected by the SAS are <i>service</i> , <i>container</i> , and <i>object</i> .
<i>services</i>	b	The service affected by the SAS is the <i>blob</i> service.

Now that the SAS is generated, copy the return value and save it somewhere for use in an upcoming step. If you generated your SAS using a method other than the Azure CLI, you will need to remove the initial `?`  if it is present. This character is a URL separator that is already provided in the URL template later in this topic where the SAS is used.

#### IMPORTANT

In production, always pass SAS tokens using TLS. Also, SAS tokens should be generated on the server and sent to the HTML page in order pass back to Azure Blob Storage. One approach you may consider is to use a serverless function to generate SAS tokens. The Azure Portal includes function templates that feature the ability to generate a SAS with a JavaScript function.

## Implement the HTML page

In this section, you'll create a basic web page and configure VS Code to launch and debug the page. Before you can launch, however, you'll need to use Node.js to start a local web server and serve the page when your browser requests it. Next, you'll add JavaScript code to call various blob storage APIs and display the results in the page. You can also see the results of these calls in the [Azure portal](#), [Azure Storage Explorer](#), and the [Azure Storage extension](#) for VS Code.

### Set up the web application

First, create a new folder named `azure-blobs-javascript` and open it in VS Code. Then create a new file in VS Code, add the following HTML, and save it as `index.html` in the `azure-blobs-javascript` folder.

```
<!DOCTYPE html>
<html>

 <body>
 <button id="create-container-button">Create container</button>
 <button id="delete-container-button">Delete 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>
 <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>

 <!-- You'll add code here later in this quickstart. -->

</html>
```

### Configure the debugger

To set up the debugger extension in VS Code, select **Debug > Add Configuration...**, then select **Chrome** or **Edge**, depending on which extension you installed in the Prerequisites section earlier. This action creates a `launch.json` file and opens it in the editor.

Next, modify the `launch.json` file so that the `url` value includes `/index.html` as shown:

```
{
 // Use IntelliSense to learn about possible attributes.
 // Hover to view descriptions of existing attributes.
 // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
 "version": "0.2.0",
 "configurations": [
 {
 "type": "chrome",
 "request": "launch",
 "name": "Launch Chrome against localhost",
 "url": "http://localhost:8080/index.html",
 "webRoot": "${workspaceFolder}"
 }
]
}
```

This configuration tells VS Code which browser to launch and which URL to load.

### Launch the web server

To launch the local Node.js web server, select **View > Terminal** to open a console window inside VS Code, then enter the following command.

```
npx http-server
```

This command will install the *http-server* package and launch the server, making the current folder available through default URLs including the one indicated in the previous step.

### Start debugging

To launch *index.html* in the browser with the VS Code debugger attached, select **Debug > Start Debugging** or press F5 in VS Code.

The UI displayed doesn't do anything yet, but you'll add JavaScript code in the following section to implement each function shown. You can then set breakpoints and interact with the debugger when it's paused on your code.

When you make changes to *index.html*, be sure to reload the page to see the changes in the browser. In VS Code, you can also select **Debug > Restart Debugging** or press CTRL + SHIFT + F5.

### Add the blob storage client library

To enable calls to the blob storage API, first [Download the Azure Storage SDK for JavaScript - Blob client library](#), extract the contents of the zip, and place the *azure-storage-blob.js* file in the *azure-blobs-javascript* folder.

Next, paste the following HTML into *index.html* after the `</body>` closing tag, replacing the placeholder comment.

```
<script src="azure-storage-blob.js" charset="utf-8"></script>

<script>
// You'll add code here in the following sections.
</script>
```

This code adds a reference to the script file and provides a place for your own JavaScript code. For the purposes of this quickstart, we're using the *azure-storage-blob.js* script file so that you can open it in VS Code, read its contents, and set breakpoints. In production, you should use the more compact *azure-storage.blob.min.js* file that is also provided in the zip file.

You can find out more about each blob storage function in the [reference documentation](#). Note that some of the functions in the SDK are only available in Node.js or only available in the browser.

The code in `azure-storage-blob.js` exports a global variable called `azblob`, which you'll use in your JavaScript code to access the blob storage APIs.

## Add the initial JavaScript code

Next, paste the following code into the `<script>` element shown in the previous code block, replacing the placeholder comment.

```
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 creates fields for each HTML element that the following code will use, and implements a `reportStatus` function to display output.

In the following sections, add each new block of JavaScript code after the previous block.

## Add your storage account info

Next, add code to access your storage account, replacing the placeholders with your account name and the SAS you generated in a previous step.

```
const accountName = "<Add your storage account name>";
const sasString = "<Add the SAS you generated earlier>";
const containerName = "testcontainer";
const containerURL = new azblob.ContainerURL(
 `https://${accountName}.blob.core.windows.net/${containerName}?${sasString}`,
 azblob.StorageURL.newPipeline(new azblob.AnonymousCredential));
```

This code uses your account info and SAS to create a `ContainerURL` instance, which is useful for creating and manipulating a storage container.

## Create and delete a storage container

Next, add code to create and delete the storage container when you press the corresponding button.

```

const createContainer = async () => {
 try {
 reportStatus(`Creating container "${containerName}"...`);
 await containerURL.create(azblob.Aborter.none);
 reportStatus(`Done.`);
 } catch (error) {
 reportStatus(error.body.message);
 }
};

const deleteContainer = async () => {
 try {
 reportStatus(`Deleting container "${containerName}"...`);
 await containerURL.delete(azblob.Aborter.none);
 reportStatus(`Done.`);
 } catch (error) {
 reportStatus(error.body.message);
 }
};

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

```

This code calls the ContainerURL [create](#) and [delete](#) functions without using an [Aborter](#) instance. To keep things simple for this quickstart, this code assumes that your storage account has been created and is enabled. In production code, use an Aborter instance to add timeout functionality.

## List blobs

Next, add code to list the contents of the storage container when you press the **List files** button.

```

const listFiles = async () => {
 fileList.size = 0;
 fileList.innerHTML = "";
 try {
 reportStatus("Retrieving file list...");
 let marker = undefined;
 do {
 const listBlobsResponse = await containerURL.listBlobFlatSegment(
 azblob.Aborter.none, marker);
 marker = listBlobsResponse.nextMarker;
 const items = listBlobsResponse.segment.blobItems;
 for (const blob of items) {
 fileList.size += 1;
 fileList.innerHTML += `<option>${blob.name}</option>`;
 }
 } while (marker);
 if (fileList.size > 0) {
 reportStatus("Done.");
 } else {
 reportStatus("The container does not contain any files.");
 }
 } catch (error) {
 reportStatus(error.body.message);
 }
};

listButton.addEventListener("click", listFiles);

```

This code calls the [ContainerURL.listBlobFlatSegment](#) function in a loop to ensure that all segments are retrieved. For each segment, it loops over the list of blob items it contains and updates the **Files** list.

## Upload blobs

Next, add code to upload files to the storage container when you press the **Select and upload files** button.

```
const uploadFiles = async () => {
 try {
 reportStatus("Uploading files...");
 const promises = [];
 for (const file of inputFile.files) {
 const blockBlobURL = azblob.BlockBlobURL.fromContainerURL(containerURL, file.name);
 promises.push(azblob.uploadBrowserDataToBlockBlob(
 azblob.Aborter.none, file, blockBlobURL));
 }
 await Promise.all(promises);
 reportStatus("Done.");
 listFiles();
 } catch (error) {
 reportStatus(error.body.message);
 }
}

selectButton.addEventListener("click", () => inputFile.click());
fileInput.addEventListener("change", uploadFiles);
```

This code connects the **Select and upload files** button to the hidden `file-input` element. In this way, 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 calls the browser-only `uploadBrowserDataToBlockBlob` function for each file you selected. Each call returns a Promise, which is added to a list so that they can all be awaited at once, causing the files to upload in parallel.

## Delete blobs

Next, add code to delete files from the storage container when you press the **Delete selected files** button.

```
const deleteFiles = async () => {
 try {
 if (fileList.selectedOptions.length > 0) {
 reportStatus("Deleting files...");
 for (const option of fileList.selectedOptions) {
 const blobURL = azblob.BlobURL.fromContainerURL(containerURL, option.text);
 await blobURL.delete(azblob.Aborter.none);
 }
 reportStatus("Done.");
 listFiles();
 } else {
 reportStatus("No files selected.");
 }
 } catch (error) {
 reportStatus(error.body.message);
 }
};

deleteButton.addEventListener("click", deleteFiles);
```

This code calls the `BlobURL.delete` 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 and test the web application

At this point, you can launch the page and experiment to get a feel for how blob storage works. If any errors occur (for example, when you try to list files before you've created the container), the **Status** pane will display the error message received. You can also set breakpoints in the JavaScript code to examine the values returned by the storage APIs.

## Clean up resources

To clean up the resources created during this quickstart, go to the [Azure portal](#) and delete the resource group you created in the Prerequisites section.

## Next steps

In this quickstart, you've created a simple website that accesses blob storage from browser-based JavaScript. To learn how you can host a website itself on blob storage, continue to the following tutorial:

[Host a static website on Blob Storage](#)

# Quickstart: Upload, download, and list blobs using Go

8/1/2019 • 8 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 additional prerequisites installed:

- [Go 1.8 or above](#)
- [Azure Storage Blob SDK for Go](#), using the following command:

```
go get -u github.com/Azure/azure-storage-blob-go/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.

## 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

This solution requires your storage account name and key to be securely stored in environment variables local to the machine running the sample. Follow one of the examples below depending on your operating System to create the environment variables.

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

```
export AZURE_STORAGE_ACCOUNT=<youraccountname>
export AZURE_STORAGE_ACCESS_KEY=<youraccountkey>
```

## Run the sample

This sample creates a test file in the current folder, uploads the test file to Blob storage, lists the blobs in the container, and downloads the file into a buffer.

To run the sample, issue the following 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-5568059279520899415
Creating a dummy file to test the upload and download
Uploading the file with blob name: 630910657703031215
Blob name: 630910657703031215
Downloaded the blob: hello world
this is a blob
Press the enter key to delete the sample files, example container, and exit the application.
```

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

The first thing to do is to 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.

- Use [SharedKeyCredential](#) struct to store your credentials.
- Create a [Pipeline](#) using the credentials and options. The pipeline specifies things like retry policies, logging, deserialization of HTTP response payloads, and more.
- Instantiate a new [ContainerURL](#), and a new [BlobURL](#) 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]**.

```
// From the Azure portal, get your storage account name and key and set environment variables.
accountName, accountKey := os.Getenv("AZURE_STORAGE_ACCOUNT"), os.Getenv("AZURE_STORAGE_ACCESS_KEY")
if len(accountName) == 0 || len(accountKey) == 0 {
 log.Fatal("Either the AZURE_STORAGE_ACCOUNT or AZURE_STORAGE_ACCESS_KEY environment variable is not set")
}

// Create a default request pipeline using your storage account name and account key.
credential, err := azblob.NewSharedKeyCredential(accountName, accountKey)
if err != nil {
 log.Fatal("Invalid credentials with error: " + err.Error())
}
p := azblob.NewPipeline(credential, azblob.PipelineOptions{})

// Create a random string for the quick start container
containerName := fmt.Sprintf("quickstart-%s", randomString())

// From the Azure portal, get your storage account blob service URL endpoint.
URL, _ := url.Parse(
 fmt.Sprintf("https://%s.blob.core.windows.net/%s", accountName, containerName))

// Create a ContainerURL object that wraps the container URL and a request
// pipeline to make requests.
containerURL := azblob.NewContainerURL(*URL, p)

// Create the container
fmt.Printf("Creating a container named %s\n", containerName)
ctx := context.Background() // This example uses a never-expiring context
_, err = containerURL.Create(ctx, azblob.Metadata{}, azblob.PublicAccessNone)
handleErrors(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.

To upload a file to a blob, open the file using `os.Open`. You can then upload the file to the specified path using one of the REST APIs: Upload (PutBlob), StageBlock/CommitBlockList (PutBlock/PutBlockList).

Alternatively, the SDK offers [high-level APIs](#) that are built on top of the low-level REST APIs. As an example, [UploadFileToBlockBlob](#) 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 **quickstartblobs-[randomstring]**.

```
// Create a file to test the upload and download.
fmt.Printf("Creating a dummy file to test the upload and download\n")
data := []byte("hello world this is a blob\n")
fileName := randomString()
err = ioutil.WriteFile(fileName, data, 0700)
handleErrors(err)

// Here's how to upload a blob.
blobURL := containerURL.NewBlockBlobURL(fileName)
file, err := os.Open(fileName)
handleErrors(err)

// You can use the low-level Upload (PutBlob) API to upload files. Low-level APIs are simple wrappers for the
// Azure Storage REST APIs.
// Note that Upload can upload up to 256MB data in one shot. Details:
// https://docs.microsoft.com/rest/api/storageservices/put-blob
// To upload more than 256MB, use StageBlock (PutBlock) and CommitBlockList (PutBlockList) functions.
// Following is commented out intentionally because we will instead use UploadFileToBlockBlob API to upload the
// blob
// _, err = blobURL.Upload(ctx, file, azblob.BlobHTTPHeaders{ContentType: "text/plain"}, azblob.Metadata{}, azblob.BlobAccessConditions{})
// handleErrors(err)

// The high-level API UploadFileToBlockBlob function uploads blocks in parallel for optimal performance, and
// can handle large files as well.
// This function calls StageBlock/CommitBlockList for files larger 256 MBs, and calls Upload for any file
// smaller
fmt.Printf("Uploading the file with blob name: %s\n", fileName)
_, err = azblob.UploadFileToBlockBlob(ctx, file, blobURL, azblob.UploadToBlockBlobOptions{
 BlockSize: 4 * 1024 * 1024,
 Parallelism: 16})
handleErrors(err)
```

## List the blobs in a container

Get a list of files in the container using the **ListBlobs** method on a **ContainerURL**. **ListBlobs** returns a single segment of blobs (up to 5000) starting from the specified **Marker**. Use an empty Marker to start enumeration from the beginning. Blob names are returned in lexicographic order. After getting a segment, process it, and then call **ListBlobs** again passing the previously returned Marker.

```
// List the container that we have created above
fmt.Println("Listing the blobs in the container:")
for marker := (azblob.Marker{}); marker.NotDone(); {
 // Get a result segment starting with the blob indicated by the current Marker.
 listBlob, err := containerURL.ListBlobsFlatSegment(ctx, marker, azblob.ListBlobsSegmentOptions{})
 handleErrors(err)

 // ListBlobs returns the start of the next segment; you MUST use this to get
 // the next segment (after processing the current result segment).
 marker = listBlob.NextMarker

 // Process the blobs returned in this result segment (if the segment is empty, the loop body won't execute)
 for _, blobInfo := range listBlob.Segment.BlobItems {
 fmt.Print(" Blob name: " + blobInfo.Name + "\n")
 }
}
```

## Download the blob

Download blobs using the **Download** low-level function on a **BlobURL**. This 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 additional 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.

```
// Here's how to download the blob
downloadResponse, err := blobURL.Download(ctx, 0, azblob.CountToEnd, azblob.BlobAccessConditions{}, false)

// NOTE: automatically retries are performed if the connection fails
bodyStream := downloadResponse.Body(azblob.RetryReaderOptions{MaxRetryRequests: 20})

// read the body into a buffer
downloadedData := bytes.Buffer{}
_, err = downloadedData.ReadFrom(bodyStream)
handleErrors(err)
```

## 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 container and the file created locally
fmt.Printf("Press enter key to delete the sample files, example container, and exit the application.\n")
bufio.NewReader(os.Stdin).ReadBytes('\n')
fmt.Printf("Cleaning up.\n")
containerURL.Delete(ctx, azblob.ContainerAccessConditions{})
file.Close()
os.Remove(fileName)
```

## Resources for developing Go applications with blobs

See these additional 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

8/1/2019 • 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.

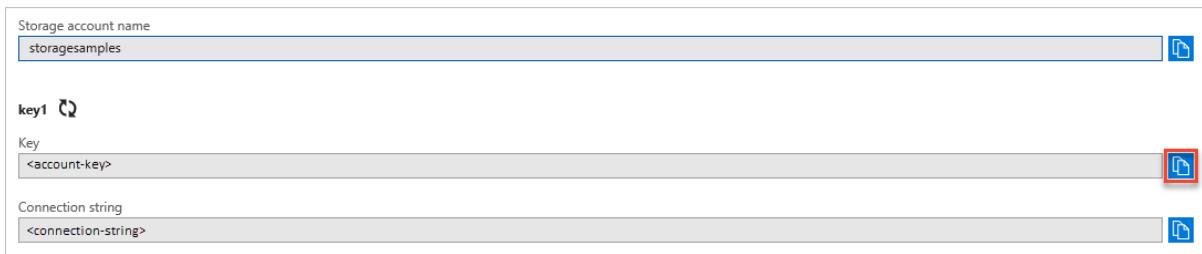
```
git clone https://github.com/Azure-Samples/storage-blobs-php-quickstart.git
```

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/blockblobsleqvxd>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: Upload, download, and list blobs using Ruby

8/1/2019 • 6 minutes to read • [Edit Online](#)

In this quickstart, you learn how to use Ruby 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:

- [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.

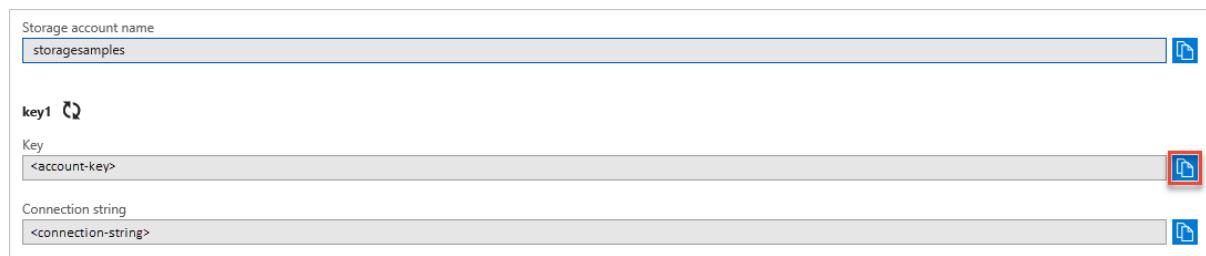
```
git clone https://github.com/Azure-Samples/storage-blobs-ruby-quickstart.git
```

This command clones the repository to your local git folder. To open the Ruby sample application, look for the storage-blobs-ruby-quickstart folder, and open the example.rb 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 `BlobService` instance for your application. Open the `example.rb` file from the Solution Explorer in your IDE. Replace the `accountname` and `accountkey` values with your account name and key.

```
blob_client = Azure::Storage::Blob::BlobService.create(
 storage_account_name: account_name,
 storage_access_key: account_key
)
```

## Run the sample

This sample creates a test file in the 'Documents' 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:

```
Creating a container: quickstartblobs7b278be3-a0dd-438b-b9cc-473401f0c0e8
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
```

When you press any key to continue, the sample program deletes the storage container and the files. Before you continue, check your 'Documents' 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 `BlobService` object to set up connection credentials.
- 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 Cloud Blob container, you can create the **Block** blob object that points to the specific blob in which you are interested, 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 set up an instance of Azure storage client, instantiate the blob service object, create a new container, and then set permissions on the container so the blobs are public. The container is called **quickstartblobs**.

```
Create a BlobService object
blob_client = Azure::Storage::Blob::BlobService.create(
 storage_account_name: account_name,
 storage_access_key: account_key
)

Create a container called 'quickstartblobs'.
container_name = 'quickstartblobs' + SecureRandom.uuid
container = blob_client.create_container(container_name)

Set the permission so the blobs are public.
blob_client.set_container_acl(container_name, "container")
```

#### 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 **create\_block\_blob()** method.

The sample code creates a local file to be used for the upload and download, storing the file to be uploaded as **file\_path\_to\_file** and the name of the blob as **local\_file\_name**. The following example uploads the file to your container called **quickstartblobs**.

```
Create a file in Documents to test the upload and download.
local_path=File.expand_path("~/Documents")
local_file_name ="QuickStart_" + SecureRandom.uuid + ".txt"
full_path_to_file =File.join(local_path, local_file_name)

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

puts "Temp file = " + full_path_to_file
puts "\nUploading to Blob storage as blob" + local_file_name

Upload the created file, using local_file_name for the blob name
blob_client.create_block_blob(container.name, local_file_name, full_path_to_file)
```

To perform a partial update of the content of a block blob, use the **create\_block\_list()** 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 `list_blobs()` method. The following code retrieves the list of blobs, then loops through them, showing the names of the blobs found in a container.

```
List the blobs in the container
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 the blobs

Download blobs to your local disk using the `get_blob()` method. The following code downloads the blob uploaded in a previous section. "\_DOWNLOADED" is added as a suffix to the blob name so you can see both files on local disk.

```
Download the blob(s).
Add '_DOWNLOADED' as prefix to '.txt' so you can see both files in Documents.
full_path_to_file2 = File.join(local_path, local_file_name.gsub('.txt', '_DOWNLOADED.txt'))

puts "\n Downloading blob to " + full_path_to_file2
blob, content = blob_client.get_blob(container_name, local_file_name)
File.open(full_path_to_file2, "wb") {|f| f.write(content)}
```

## 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. If the files created are no longer needed, you use the `delete_blob()` method to delete the files.

```
Clean up resources. This includes the container and the temp files
blob_client.delete_container(container_name)
File.delete(full_path_to_file)
File.delete(full_path_to_file2)
```

# Resources for developing Ruby applications with blobs

See these additional resources for Ruby development with Blob storage:

- View and download the [Ruby client library source code](#) for Azure Storage on GitHub.
- Explore [Blob storage samples](#) written using the Ruby client library.

## Next steps

In this quickstart, you learned how to transfer files between a local disk and Azure blob storage using Ruby. To learn more about working with blob storage, continue to the Blob storage How-to.

### [Blob Storage Operations How-To](#)

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



# Tutorial: Upload image data in the cloud with Azure Storage

4/15/2020 • 9 minutes to read • [Edit Online](#)

This tutorial is part one of a series. In this tutorial, you will learn how to deploy a web app that uses the Azure Blob storage client library to upload images to a storage account. When you're finished, you'll have a web app that stores and displays images from Azure storage.

- [.NET v12 SDK](#)
- [Node.js v10 SDK](#)

The screenshot shows a web browser window with the following details:

- Title Bar:** Home Page - ImageRes X
- Address Bar:** contosowebapp123qwe.azurewebsites.net
- Content Area:**
  - Section Header:** ImageResizer
  - Form:** A dashed blue-bordered area with the text "Drop files here or click to upload."
  - Section Header:** Generated Thumbnails
  - Thumbnail Image:** A small image of wind turbines.
  - Note:** This app has no official privacy policy. Your data will be uploaded to a service in order produce a picture. Your images will be public once you upload them and there is no automated way to remove them.

In part one of the series, you learn how to:

- Create a storage account
- Create a container and set permissions
- Retrieve an access key
- Deploy a web app to Azure
- Configure app settings
- Interact with the web app

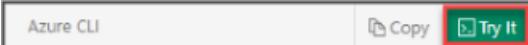
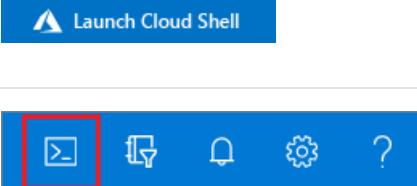
# Prerequisites

To complete this tutorial, you need an Azure subscription. Create a [free account](#) before you begin.

## 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 <a href="#">Azure portal</a> .	

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.

To install and use the CLI locally, this tutorial requires that you run the Azure CLI version 2.0.4 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install the Azure CLI](#).

## Create a resource group

Create a resource group with the `az group create` command. An Azure resource group is a logical container into which Azure resources are deployed and managed.

The following example creates a resource group named `myResourceGroup`.

```
az group create --name myResourceGroup --location southeastasia
```

## Create a storage account

The sample uploads images 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 `az storage account create` command.

## IMPORTANT

In part 2 of the tutorial, you use Azure Event Grid with Blob storage. Make sure to create your storage account in an Azure region that supports Event Grid. For a list of supported regions, see [Azure products by region](#).

In the following command, replace your own globally unique name for the Blob storage account where you see the `<blob_storage_account>` placeholder.

```
blobStorageAccount=<blob_storage_account>

az storage account create --name $blobStorageAccount --location southeastasia \
--resource-group myResourceGroup --sku Standard_LRS --kind StorageV2 --access-tier hot
```

## Create Blob storage containers

The app uses two containers in the Blob storage account. Containers are similar to folders and store blobs. The *images* container is where the app uploads full-resolution images. In a later part of the series, an Azure function app uploads resized image thumbnails to the *thumbnails* container.

Get the storage account key by using the [az storage account keys list](#) command. Then, use this key to create two containers with the [az storage container create](#) command.

The *images* container's public access is set to `off`. The *thumbnails* container's public access is set to `container`. The `container` public access setting permits users who visit the web page to view the thumbnails.

```
blobStorageAccountKey=$(az storage account keys list -g myResourceGroup \
-n $blobStorageAccount --query "[0].value" --output tsv)

az storage container create -n images --account-name $blobStorageAccount \
--account-key $blobStorageAccountKey --public-access off

az storage container create -n thumbnails --account-name $blobStorageAccount \
--account-key $blobStorageAccountKey --public-access container

echo "Make a note of your Blob storage account key..."
echo $blobStorageAccountKey
```

Make a note of your Blob storage account name and key. The sample app uses these settings to connect to the storage account to upload the images.

## Create an App Service plan

An [App Service plan](#) specifies the location, size, and features of the web server farm that hosts your app.

Create an App Service plan with the [az appservice plan create](#) command.

The following example creates an App Service plan named `myAppServicePlan` in the **Free** pricing tier:

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku Free
```

## Create a web app

The web app provides a hosting space for the sample app code that's deployed from the GitHub sample repository. Create a [web app](#) in the `myAppServicePlan` App Service plan with the [az webapp create](#) command.

In the following command, replace `<web_app>` with a unique name. Valid characters are `a-z`, `0-9`, and `-`. If `<web_app>` is not unique, you get the error message: *Website with given name <web\_app> already exists.* The default URL of the web app is [https://<web\\_app>.azurewebsites.net](https://<web_app>.azurewebsites.net).

```
webapp=<web_app>

az webapp create --name $webapp --resource-group myResourceGroup --plan myAppServicePlan
```

## Deploy the sample app from the GitHub repository

- [.NET v12 SDK](#)
- [Node.js v10 SDK](#)

App Service supports several ways to deploy content to a web app. In this tutorial, you deploy the web app from a [public GitHub sample repository](#). Configure GitHub deployment to the web app with the [az webapp deployment source config](#) command.

The sample project contains an [ASP.NET MVC](#) app. The app accepts an image, saves it to a storage account, and displays images from a thumbnail container. The web app uses the [Azure.Storage](#), [Azure.Storage.Blobs](#), and [Azure.Storage.Blobs.Models](#) namespaces to interact with the Azure Storage service.

```
az webapp deployment source config --name $webapp --resource-group myResourceGroup \
--branch master --manual-integration \
--repo-url https://github.com/Azure-Samples/storage-blob-upload-from-webapp
```

## Configure web app settings

- [.NET v12 SDK](#)
- [Node.js v10 SDK](#)

The sample web app uses the [Azure Storage APIs for .NET](#) to upload images. Storage account credentials are set in the app settings for the web app. Add app settings to the deployed app with the [az webapp config appsettings set](#) command.

```
az webapp config appsettings set --name $webapp --resource-group myResourceGroup \
--settings AzureStorageConfig_AccountName=$blobStorageAccount \
AzureStorageConfig_ImageContainer=images \
AzureStorageConfig_ThumbnailContainer=thumbnails \
AzureStorageConfig_AccountKey=$blobStorageAccountKey
```

After you deploy and configure the web app, you can test the image upload functionality in the app.

## Upload an image

To test the web app, browse to the URL of your published app. The default URL of the web app is [https://<web\\_app>.azurewebsites.net](https://<web_app>.azurewebsites.net).

- [.NET v12 SDK](#)
- [Node.js v10 SDK](#)

Select the **Upload photos** region to specify and upload a file, or drag a file onto the region. The image disappears if successfully uploaded. The **Generated Thumbnails** section will remain empty until we test it later in this topic.

The screenshot shows a web browser window with the title "Home Page - ImageRes" and the URL "contosowebapp123qwe.azurewebsites.net". The page has a header with the "ImageResizer" logo. Below the header, there's a section titled "Upload photos" with a dashed blue border and a placeholder text "Drop files here or click to upload.". Underneath this, there's a section titled "Generated Thumbnails" which currently displays the message "This app has no official privacy policy. Your data will be uploaded to a service in order produce a picture. Your images will be public once you upload them and there is no automated way to remove them."

In the sample code, the `UploadFileToStorage` task in the `Storagehelper.cs` file is used to upload the images to the `images` container within the storage account using the `UploadAsync` method. The following code sample contains the `UploadFileToStorage` task.

```
public static async Task<bool> UploadFileToStorage(Stream fileStream, string fileName,
 AzureStorageConfig _storageConfig)
{
 // Create a URI to the blob
 Uri blobUri = new Uri("https://" +
 _storageConfig.AccountName +
 ".blob.core.windows.net/" +
 _storageConfig.ImageContainer +
 "/" + fileName);

 // Create StorageSharedKeyCredentials object by reading
 // the values from the configuration (appsettings.json)
 StorageSharedKeyCredential storageCredentials =
 new StorageSharedKeyCredential(_storageConfig.AccountName, _storageConfig.AccountKey);

 // Create the blob client.
 BlobClient blobClient = new BlobClient(blobUri, storageCredentials);

 // Upload the file
 await blobClient.UploadAsync(fileStream);

 return await Task.FromResult(true);
}
```

The following classes and methods are used in the preceding task:

CLASS	METHOD
<code>Uri</code>	<code>Uri constructor</code>

CLASS	METHOD
StorageSharedKeyCredential	StorageSharedKeyCredential(String, String) constructor
BlobClient	UploadAsync

## Verify the image is shown in the storage account

Sign in to the [Azure portal](#). From the left menu, select **Storage accounts**, then select the name of your storage account. Select **Containers**, then select the **images** container.

Verify the image is shown in the container.

NAME	MODIFIED	ACCESS TIER	BLOB TYPE	SIZE	LEASE STATE
WindFar...	9/11/2017 12:06:27 PM	Default	Block blob	204.58 KiB	Available

## Test thumbnail viewing

To test thumbnail viewing, you'll upload an image to the **thumbnails** container to check whether the app can read the **thumbnails** container.

Sign in to the [Azure portal](#). From the left menu, select **Storage accounts**, then select the name of your storage account. Select **Containers**, then select the **thumbnails** container. Select **Upload** to open the **Upload blob** pane.

Choose a file with the file picker and select **Upload**.

Navigate back to your app to verify that the image uploaded to the **thumbnails** container is visible.

- [.NET v12 SDK](#)
- [Node.js v10 SDK](#)

The screenshot shows a web browser window with the title "Home Page - ImageResizer" and the URL "contosowebapp123qwe.azurewebsites.net". The page content includes a section titled "Upload photos" with a dashed blue border and a placeholder text "Drop files here or click to upload.". Below this is a section titled "Generated Thumbnails" containing a thumbnail image of wind turbines. At the bottom, a note states: "This app has no official privacy policy. Your data will be uploaded to a service in order produce a picture. Your images will be public once you upload them and there is no automated way to remove them."

In part two of the series, you automate thumbnail image creation so you don't need this image. In the **thumbnails** container in the Azure portal, select the image you uploaded and select **Delete** to delete the image.

You can enable Content Delivery Network (CDN) to cache content from your Azure storage account. For more information about how to enable CDN with your Azure storage account, see [Integrate an Azure storage account with Azure CDN](#).

## Next steps

In part one of the series, you learned how to configure a web app to interact with storage.

Go on to part two of the series to learn about using Event Grid to trigger an Azure function to resize an image.

[Use Event Grid to trigger an Azure Function to resize an uploaded image](#)

# Tutorial: Automate resizing uploaded images using Event Grid

4/16/2020 • 8 minutes to read • [Edit Online](#)

Azure Event Grid is an eventing service for the cloud. Event Grid enables you to create subscriptions to events raised by Azure services or third-party resources.

This tutorial is part two of a series of Storage tutorials. It extends the [previous Storage tutorial](#) to add serverless automatic thumbnail generation using Azure Event Grid and Azure Functions. Event Grid enables [Azure Functions](#) to respond to [Azure Blob storage](#) events and generate thumbnails of uploaded images. An event subscription is created against the Blob storage create event. When a blob is added to a specific Blob storage container, a function endpoint is called. Data passed to the function binding from Event Grid is used to access the blob and generate the thumbnail image.

You use the Azure CLI and the Azure portal to add the resizing functionality to an existing image upload app.

- [.NET v12 SDK](#)
- [Node.js V10 SDK](#)

The screenshot shows a web browser window titled "Home Page - ImageRe: X". The main content area is titled "ImageResizer". Below it, a section titled "Upload photos" contains a dashed blue rectangular area with the text "Drop files here or click to upload.". Below this, a section titled "Generated Thumbnails" displays a thumbnail image of a person sitting at a desk with a laptop. At the bottom of the page, a note states: "This app has no official privacy policy. Your data will be uploaded to a service in order to produce a picture. Your images will be public once you upload them and there is no automated way to remove them."

## Generated Thumbnails



This app has no official privacy policy. Your data will be uploaded to a service in order to produce a picture. Your images will be public once you upload them and there is no automated way to remove them.

In this tutorial, you learn how to:

- Create an Azure Storage account
- Deploy serverless code using Azure Functions
- Create a Blob storage event subscription in Event Grid

## Prerequisites

### NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

To complete this tutorial:

You must have completed the previous Blob storage tutorial: [Upload image data in the cloud with Azure Storage](#).

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

# 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 <a href="#">Azure portal</a> .	

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 CLI locally, this tutorial requires the Azure CLI version 2.0.14 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

If you are not using Cloud Shell, you must first sign in using `az login`.

If you've not previously registered the Event Grid resource provider in your subscription, make sure it's registered.

```
az provider register --namespace Microsoft.EventGrid
```

## Create an Azure Storage account

Azure Functions requires a general storage account. In addition to the Blob storage account you created in the previous tutorial, create a separate general storage account in the resource group by using the `az storage account create` command. Storage account names must be between 3 and 24 characters in length and may contain numbers and lowercase letters only.

1. Set a variable to hold the name of the resource group that you created in the previous tutorial.

```
resourceGroupName="myResourceGroup"
```

2. Set a variable for the name of the new storage account that Azure Functions requires.

```
functionstorage="<name of the storage account to be used by the function>"
```

### 3. Create the storage account for the Azure function.

```
az storage account create --name $functionstorage --location southeastasia \
--resource-group $resourceGroupName --sku Standard_LRS --kind StorageV2
```

## Create a function app

You must have a function app to host the execution of your function. The function app provides an environment for serverless execution of your function code. Create a function app by using the [az functionapp create](#) command.

In the following command, provide your own unique function app name. The function app name is used as the default DNS domain for the function app, and so the name needs to be unique across all apps in Azure.

1. Specify a name for the function app that's to be created.

```
functionapp=<name of the function app>
```

2. Create the Azure function.

```
az functionapp create --name $functionapp --storage-account $functionstorage \
--resource-group $resourceGroupName --consumption-plan-location southeastasia \
--functions-version 2
```

Now configure the function app to connect to the Blob storage account you created in the [previous tutorial](#).

## Configure the function app

The function needs credentials for the Blob storage account, which are added to the application settings of the function app using the [az functionapp config appsettings set](#) command.

- [.NET v12 SDK](#)
- [Node.js V10 SDK](#)

```
blobStorageAccount=<name of the Blob storage account you created in the previous tutorial>
storageConnectionString=$(az storage account show-connection-string --resource-group $resourceGroupName \
--name $blobStorageAccount --query connectionString --output tsv)

az functionapp config appsettings set --name $functionapp --resource-group $resourceGroupName \
--settings AzureWebJobsStorage=$storageConnectionString THUMBNAIL_CONTAINER_NAME=thumbnails \
THUMBNAIL_WIDTH=100 FUNCTIONS_EXTENSION_VERSION=~2
```

The `FUNCTIONS_EXTENSION_VERSION=~2` setting makes the function app run on version 2.x of the Azure Functions runtime.

You can now deploy a function code project to this function app.

## Deploy the function code

- [.NET v12 SDK](#)
- [Node.js V10 SDK](#)

The sample C# resize function is available on [GitHub](#). Deploy this code project to the function app by using the [az functionapp deployment source config](#) command.

```
az functionapp deployment source config --name $functionapp --resource-group $resourceGroupName \
--branch master --manual-integration \
--repo-url https://github.com/Azure-Samples/function-image-upload-resize
```

The image resize function is triggered by HTTP requests sent to it from the Event Grid service. You tell Event Grid that you want to get these notifications at your function's URL by creating an event subscription. For this tutorial you subscribe to blob-created events.

The data passed to the function from the Event Grid notification includes the URL of the blob. That URL is in turn passed to the input binding to obtain the uploaded image from Blob storage. The function generates a thumbnail image and writes the resulting stream to a separate container in Blob storage.

This project uses `EventGridTrigger` for the trigger type. Using the Event Grid trigger is recommended over generic HTTP triggers. Event Grid automatically validates Event Grid Function triggers. With generic HTTP triggers, you must implement the [validation response](#).

- [.NET v12 SDK](#)
- [Node.js V10 SDK](#)

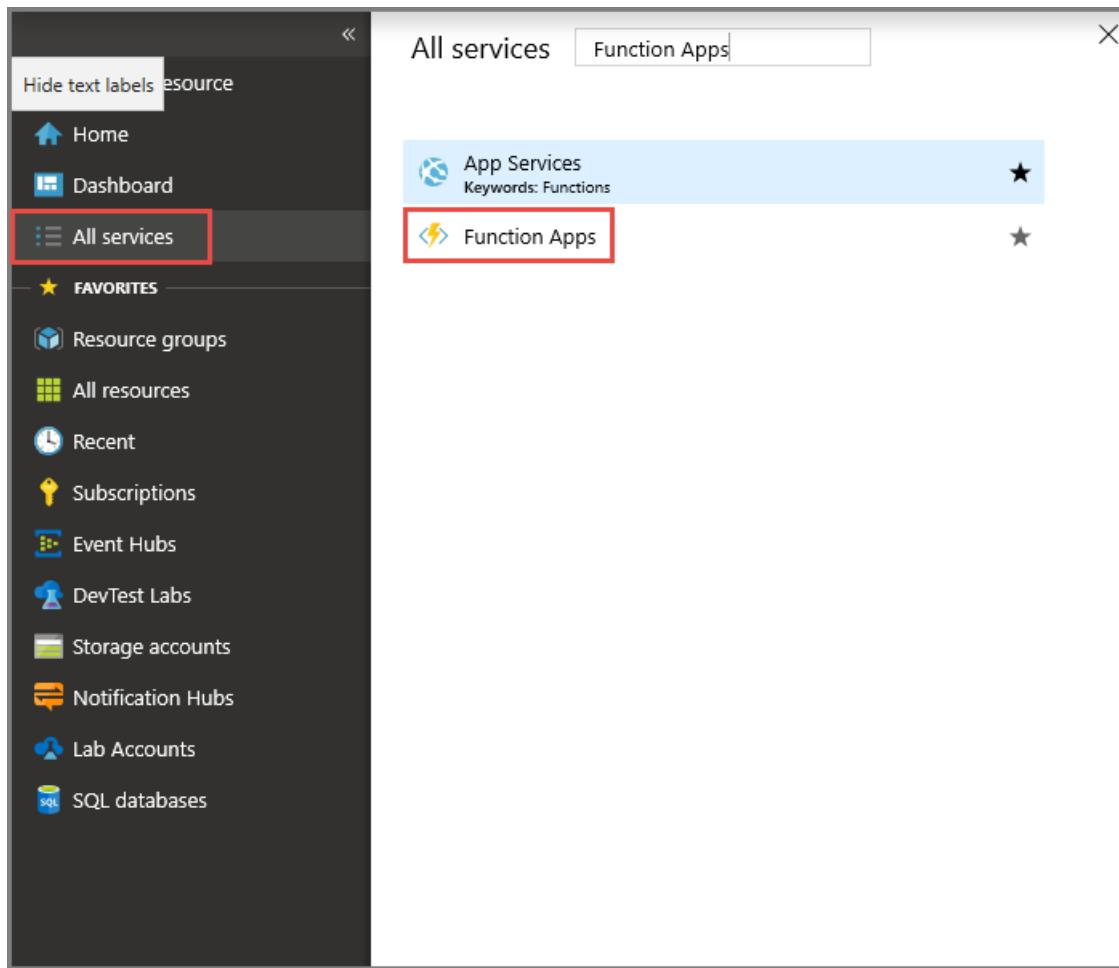
To learn more about this function, see the [function.json and run.csx files](#).

The function project code is deployed directly from the public sample repository. To learn more about deployment options for Azure Functions, see [Continuous deployment for Azure Functions](#).

## Create an event subscription

An event subscription indicates which provider-generated events you want sent to a specific endpoint. In this case, the endpoint is exposed by your function. Use the following steps to create an event subscription that sends notifications to your function in the Azure portal:

1. In the [Azure portal](#), select **All Services** on the left menu, and then select **Function Apps**.



2. Expand your function app, choose the **Thumbnail** function, and then select **Add Event Grid subscription**.

The screenshot shows the Azure portal's function details page for 'myfuncapp0130 - Thumbnail'. On the left, there's a sidebar with a search bar containing 'myfuncapp0130', a dropdown menu, and a 'Function Apps' section. Under 'Function Apps', 'myfuncapp0130' is expanded, showing 'Functions (Read Only)', 'Thumbnail' (which is selected and highlighted with a red box), 'Integrate', 'Manage', 'Monitor', 'Proxies (Read Only)', and 'Slots (preview)'. On the right, there's a preview area with a message about being in read-only mode due to source control integration. Below that is a 'function.json' editor with the following code:

```
1 "generatedBy": "Microsoft.NET.Sdk.Functions-1.0.22",
2 "configurationSource": "attributes",
3 "bindings": [
4 {
5 "type": "eventGridTrigger",
6 "name": "eventGridEvent"
7 }
8],
9 "disabled": false,
10 "scriptFile": "../bin/ImageFunctions.dll",
11 "entryPoint": "ImageFunctions.Thumbnail.Run"
12
13]
```

3. Use the event subscription settings as specified in the table.

 **Create Event Subscription**

Event Grid

Basic    Filters    Additional Features

Event Subscriptions listen for events emitted by the topic resource and send them to the endpoint resource. [Learn more](#)

**EVENT SUBSCRIPTION DETAILS**

Name	imageresizersub	
Event Schema	Event Grid Schema	

**TOPIC DETAILS**

Pick a topic resource for which events should be pushed to your destination. [Learn more](#)

Topic Type	 Storage account
Topic Resource	<a href="#">myblobstorage0401 (change)</a>

**EVENT TYPES**

Pick which event types get pushed to your destination. [Learn more](#)

Filter to Event Types	2 selected	
-----------------------	------------	-------------------------------------------------------------------------------------

**ENDPOINT DETAILS**

Pick an event handler to receive your events. [Learn more](#)

Endpoint Type	 Azure Function <a href="#">(change)</a>
Endpoint	<a href="#">Thumbnail (change)</a>

**Create**

SETTING	SUGGESTED VALUE	DESCRIPTION
Name	imageresizersub	Name that identifies your new event subscription.
Topic type	Storage accounts	Choose the Storage account event provider.
Subscription	Your Azure subscription	By default, your current Azure subscription is selected.
Resource group	myResourceGroup	Select <b>Use existing</b> and choose the resource group you have been using in this tutorial.
Resource	Your Blob storage account	Choose the Blob storage account you created.
Event types	Blob created	Uncheck all types other than <b>Blob created</b> . Only event types of <code>Microsoft.Storage.BlobCreated</code> are passed to the function.

SETTING	SUGGESTED VALUE	DESCRIPTION
Endpoint type	autogenerated	Pre-defined as <b>Azure Function</b> .
Endpoint	autogenerated	Name of the function. In this case, it's <b>Thumbnail</b> .

4. Switch to the **Filters** tab, and do the following actions:

- a. Select **Enable subject filtering** option.
- b. For **Subject begins with**, enter the following value :  
**/blobServices/default/containers/images/blobs/**.

**Create Event Subscription**  
Event Grid

**Basic**   **Filters**   **Additional Features**

**SUBJECT FILTERS**

Apply filters to the subject of each event. Only events with matching subjects get delivered. [Learn more](#)

Enable subject filtering

Subject Begins With

Subject Ends With

Case-sensitive subject matching

**ADVANCED FILTERS**

Filter on attributes of each event. Only events that match all filters get delivered. Up to 5 filters can be specified. All string comparisons are case-insensitive. [Learn more](#)

Valid keys for currently selected event schema:

- id, topic, subject, eventtype, dataversion
- Custom properties at most one level inside the data payload, using "." as the nesting separator. (e.g. data, data.key are valid, data.key.key is not)

KEY	OPERATOR	VALUE
No results		
<a href="#">Add new filter</a>		

**Create**

5. Select **Create** to add the event subscription. This creates an event subscription that triggers the **Thumbnail** function when a blob is added to the **images** container. The function resizes the images and adds them to the **thumbnails** container.

Now that the backend services are configured, you test the image resize functionality in the sample web app.

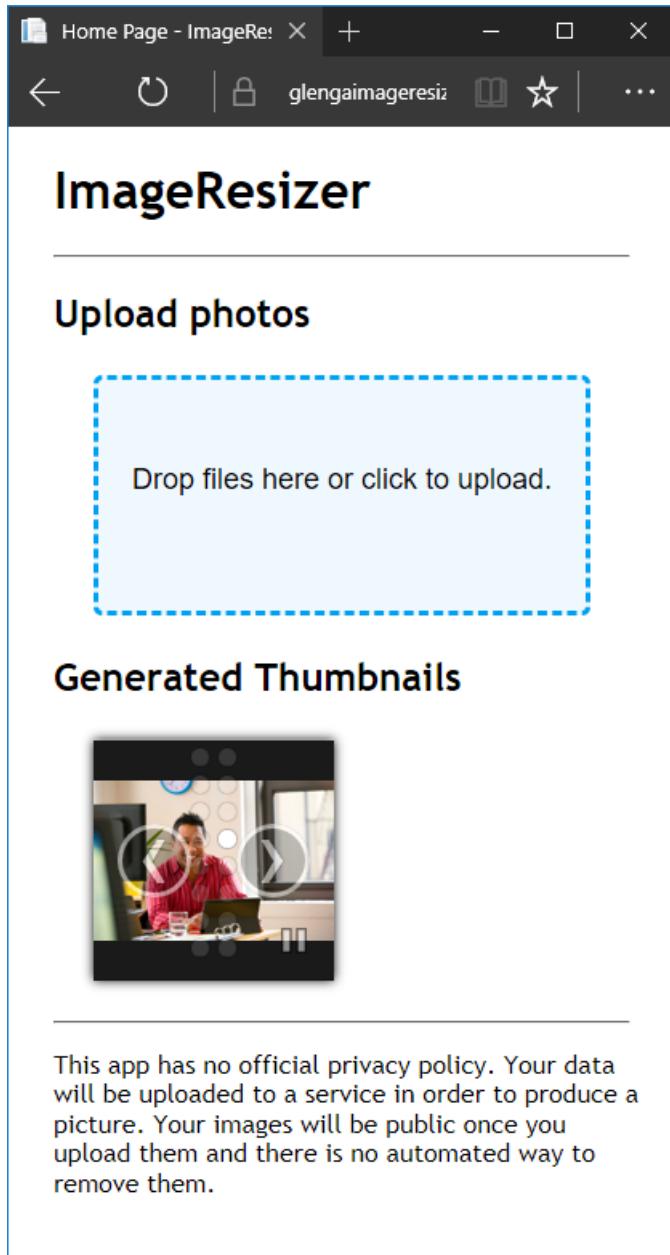
## Test the sample app

To test image resizing in the web app, browse to the URL of your published app. The default URL of the web app is [https://<web\\_app>.azurewebsites.net](https://<web_app>.azurewebsites.net).

- [.NET v12 SDK](#)
- [Node.js V10 SDK](#)

Click the **Upload photos** region to select and upload a file. You can also drag a photo to this region.

Notice that after the uploaded image disappears, a copy of the uploaded image is displayed in the **Generated Thumbnails** carousel. This image was resized by the function, added to the *thumbnails* container, and downloaded by the web client.



## Next steps

In this tutorial, you learned how to:

- Create a general Azure Storage account
- Deploy serverless code using Azure Functions
- Create a Blob storage event subscription in Event Grid

Advance to part three of the Storage tutorial series to learn how to secure access to the storage account.

### Secure access to an applications data in the cloud

- To learn more about Event Grid, see [An introduction to Azure Event Grid](#).
- To try another tutorial that features Azure Functions, see [Create a function that integrates with Azure Logic Apps](#).

# Secure access to application data

3/20/2020 • 4 minutes to read • [Edit Online](#)

This tutorial is part three of a series. You learn how to secure access to the storage account.

In part three of the series, you learn how to:

- Use SAS tokens to access thumbnail images
- Turn on server-side encryption
- Enable HTTPS-only transport

[Azure blob storage](#) provides a robust service to store files for applications. This tutorial extends [the previous topic](#) to show how to secure access to your storage account from a web application. When you're finished the images are encrypted and the web app uses secure SAS tokens to access the thumbnail images.

## Prerequisites

To complete this tutorial you must have completed the previous Storage tutorial: [Automate resizing uploaded images using Event Grid](#).

## Set container public access

In this part of the tutorial series, SAS tokens are used for accessing the thumbnails. In this step, you set the public access of the *thumbnails* container to `off`.

```
blobStorageAccount=<blob_storage_account>

blobStorageAccountKey=$(az storage account keys list -g myResourceGroup \
 --name $blobStorageAccount --query [0].value --output tsv)

az storage container set-permission \
 --account-name $blobStorageAccount \
 --account-key $blobStorageAccountKey \
 --name thumbnails \
 --public-access off
```

## Configure SAS tokens for thumbnails

In part one of this tutorial series, the web application was showing images from a public container. In this part of the series, you use shared access signatures (SAS) tokens to retrieve the thumbnail images. SAS tokens allow you to provide restricted access to a container or blob based on IP, protocol, time interval, or rights allowed. For more information about SAS, see [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#).

In this example, the source code repository uses the `sasTokens` branch, which has an updated code sample. Delete the existing GitHub deployment with the [az webapp deployment source delete](#). Next, configure GitHub deployment to the web app with the [az webapp deployment source config](#) command.

In the following command, `<web-app>` is the name of your web app.

```
az webapp deployment source delete --name <web-app> --resource-group myResourceGroup

az webapp deployment source config --name <web_app> \
--resource-group myResourceGroup --branch sasTokens --manual-integration \
--repo-url https://github.com/Azure-Samples/storage-blob-upload-from-webapp
```

The `sasTokens` branch of the repository updates the `StorageHelper.cs` file. It replaces the `GetThumbNailUrls` task with the code example below. The updated task retrieves the thumbnail URLs by using a `BlobSasBuilder` to specify the start time, expiry time, and permissions for the SAS token. Once deployed the web app now retrieves the thumbnails with a URL using a SAS token. The updated task is shown in the following example:

```
public static async Task<List<string>> GetThumbNailUrls(AzureStorageConfig _storageConfig)
{
 List<string> thumbnailUrls = new List<string>();

 // Create a URI to the storage account
 Uri accountUri = new Uri("https://" + _storageConfig.AccountName + ".blob.core.windows.net/");

 // Create BlobServiceClient from the account URI
 BlobServiceClient blobServiceClient = new BlobServiceClient(accountUri);

 // Get reference to the container
 BlobContainerClient container =
 blobServiceClient.GetBlobContainerClient(_storageConfig.ThumbnailContainer);

 if (container.Exists())
 {
 // Set the expiration time and permissions for the container.
 // In this case, the start time is specified as a few
 // minutes in the past, to mitigate clock skew.
 // The shared access signature will be valid immediately.
 BlobSasBuilder sas = new BlobSasBuilder()
 {
 Resource = "c",
 BlobContainerName = _storageConfig.ThumbnailContainer,
 StartsOn = DateTimeOffset.UtcNow.AddMinutes(-5),
 ExpiresOn = DateTimeOffset.UtcNow.AddHours(1)
 };

 sas.SetPermissions(BlobContainerSasPermissions.All);

 // Create StorageSharedKeyCredentials object by reading
 // the values from the configuration (appsettings.json)
 StorageSharedKeyCredential storageCredential =
 new StorageSharedKeyCredential(_storageConfig.AccountName, _storageConfig.AccountKey);

 // Create a SAS URI to the storage account
 UriBuilder sasUri = new UriBuilder(accountUri);
 sasUri.Query = sas.ToSasQueryParameters(storageCredential).ToString();

 foreach (BlobItem blob in container.GetBlobs())
 {
 // Create the URI using the SAS query token.
 string sasBlobUri = container.Uri + "/" +
 blob.Name + sasUri.Query;

 //Return the URI string for the container, including the SAS token.
 thumbnailUrls.Add(sasBlobUri);
 }
 }
 return await Task.FromResult(thumbnailUrls);
}
```

The following classes, properties, and methods are used in the preceding task:

CLASS	PROPERTIES	METHODS
StorageSharedKeyCredential		
BlobServiceClient		GetBlobContainerClient
BlobContainerClient	Uri	Exists GetBlobs
BlobSasBuilder		SetPermissions ToSasQueryParameters
BlobItem	Name	
UriBuilder	Query	
List		Add

## Server-side encryption

Azure Storage Service Encryption (SSE) helps you protect and safeguard your data. SSE encrypts data at rest, handling encryption, decryption, and key management. All data is encrypted using 256-bit [AES encryption](#), one of the strongest block ciphers available.

SSE automatically encrypts data in all performance tiers (Standard and Premium), all deployment models (Azure Resource Manager and Classic), and all of the Azure Storage services (Blob, Queue, Table, and File).

## Enable HTTPS only

In order to ensure that requests for data to and from a storage account are secure, you can limit requests to HTTPS only. Update the storage account required protocol by using the [az storage account update](#) command.

```
az storage account update --resource-group myresourcegroup --name <storage-account-name> --https-only true
```

Test the connection using `curl` using the `HTTP` protocol.

```
curl http://<storage-account-name>.blob.core.windows.net/<container>/<blob-name> -I
```

Now that secure transfer is required, you receive the following message:

```
HTTP/1.1 400 The account being accessed does not support http.
```

## Next steps

In part three of the series, you learned how to secure access to the storage account, such as how to:

- Use SAS tokens to access thumbnail images
- Turn on server-side encryption
- Enable HTTPS-only transport

Advance to part four of the series to learn how to monitor and troubleshoot a cloud storage application.

Monitor and troubleshoot application cloud application storage

# Monitor and troubleshoot a cloud storage application

8/7/2019 • 3 minutes to read • [Edit Online](#)

This tutorial is part four and the final part of a series. You learn how to monitor and troubleshoot a cloud storage application.

In part four of the series, you learn how to:

- Turn on logging and metrics
- Enable alerts for authorization errors
- Run test traffic with incorrect SAS tokens
- Download and analyze logs

[Azure storage analytics](#) provides logging and metric data for a storage account. This data provides insights into the health of your storage account. To collect data from Azure storage analytics, you can configure logging, metrics and alerts. This process involves turning on logging, configuring metrics, and enabling alerts.

Logging and metrics from storage accounts are enabled from the **Diagnostics** tab in the Azure portal. Storage logging enables you to record details for both successful and failed requests in your storage account. These logs enable you to see details of read, write, and delete operations against your Azure tables, queues, and blobs. They also enable you to see the reasons for failed requests such as timeouts, throttling, and authorization errors.

## Log in to the Azure portal

Log in to the [Azure portal](#)

## Turn on logging and metrics

From the left menu, select **Resource Groups**, select **myResourceGroup**, and then select your storage account in the resource list.

Under **Diagnostics settings (classic)** set **Status** to **On**. Ensure all of the options under **Blob properties** are enabled.

When complete, click **Save**

The screenshot shows the Microsoft Azure portal interface for managing diagnostic settings. On the left, a sidebar lists various service categories like TABLE SERVICE, QUEUE SERVICE, and MONITORING. Under MONITORING, the 'Diagnostic settings (classic)' link is highlighted with a red box. The main content area displays the configuration for 'mymetricsdemo - Diagnostic settings (classic)'. It includes tabs for Blob properties, File properties, Table properties, and Queue properties. The Blob properties tab is active, showing three sections: Hour metrics, Minute metrics, and Logging. Each section has checkboxes for enabling metrics and selecting specific metrics (e.g., API metrics, Delete data). Each section also includes a 'days' input field set to 7. At the top right of the configuration panel, there are Save and Discard buttons.

## Enable alerts

Alerts provide a way to email administrators or trigger a webhook based on a metric breaching a threshold. In this example, you enable an alert for the `SASClientOtherError` metric.

### Navigate to the storage account in the Azure portal

Under the Monitoring section, select **Alerts (classic)**.

Select **Add metric alert (classic)** and complete the **Add rule** form by filling in the required information. From the **Metric** dropdown, select `SASClientOtherError`. To allow your alert to trigger upon the first error, from the **Condition** dropdown select **Greater than or equal to**.

The screenshot shows the 'Add rule' dialog box. The 'Name' field is populated with 'myStorageAlertRule'. The 'Description' field contains the text 'Alert on Client Authorization Errors'. The 'Source' dropdown is set to 'Metrics'. The 'Criteria' dropdown is set to 'Mv Subscription'.

Resource group  
myResourceGroup

Resource  
mymetricsdemo/blob

\* Metric ⓘ  
SASClientOtherError

2  
1.5  
No data to display  
0.5  
0  
7 PM

Condition  
Greater than or equal to

\* Threshold  
1

Period ⓘ  
Over the last 5 minutes

Notify via

Email owners, contributors, and readers

Additional administrator email(s)  
contoso@microsoft.com

Webhook ⓘ  
HTTP or HTTPS endpoint to route alerts to

Learn more about configuring webhooks

Take action ⓘ  
Run a logic app from this alert >

OK

## Simulate an error

To simulate a valid alert, you can attempt to request a non-existent blob from your storage account. The following command requires a storage container name. You can either use the name of an existing container or create a new one for the purposes of this example.

Replace the placeholders with real values (make sure <INCORRECT\_BLOB\_NAME> is set to a value that does not exist) and run the command.

```
sasToken=$(az storage blob generate-sas \
--account-name <STORAGE_ACCOUNT_NAME> \
--account-key <STORAGE_ACCOUNT_KEY> \
--container-name <CONTAINER_NAME> \
--name <INCORRECT_BLOB_NAME> \
--permissions r \
--expiry `date --date="next day" +%Y-%m-%d`)

curl https://<STORAGE_ACCOUNT_NAME>.blob.core.windows.net/<CONTAINER_NAME>/<INCORRECT_BLOB_NAME>?$sasToken
```

The following image is an example alert that is based off the simulated failure ran with the preceding example.

Dear Customer,

**⚠ 'SASClientOtherError  
GreaterThanOrEqualTo 1 (Count) in the  
last 5 minutes' was activated for  
mymetricsdemo/blob**

You can view more details for this alert in the [Microsoft Azure Management Portal](#).

**RULE NAME:** SASClientOtherRule

**RULE DESCRIPTION:**

**SERVICE:** storageaccounts: mymetricsdemo (mydemogroup)

**METRIC:** Total SASClientOtherError

**ALERT ACTIVATED TIME (UTC):** 7/20/2018 3:22:33 PM

**SUBSCRIPTION NAME:** My Subscription

Thank you,  
Microsoft Azure Team

## Download and view logs

Storage logs store data in a set of blobs in a blob container named **\$logs** in your storage account. This container does not show up if you list all the blob containers in your account but you can see its contents if you access it directly.

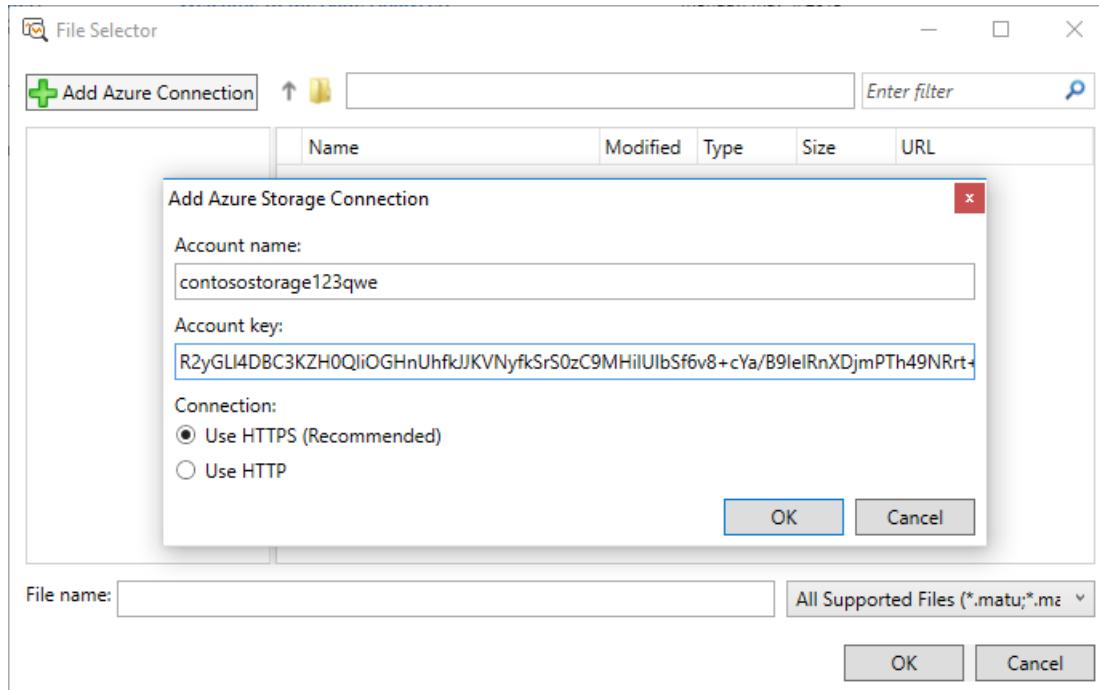
In this scenario, you use [Microsoft Message Analyzer](#) to interact with your Azure storage account.

[Download Microsoft Message Analyzer](#)

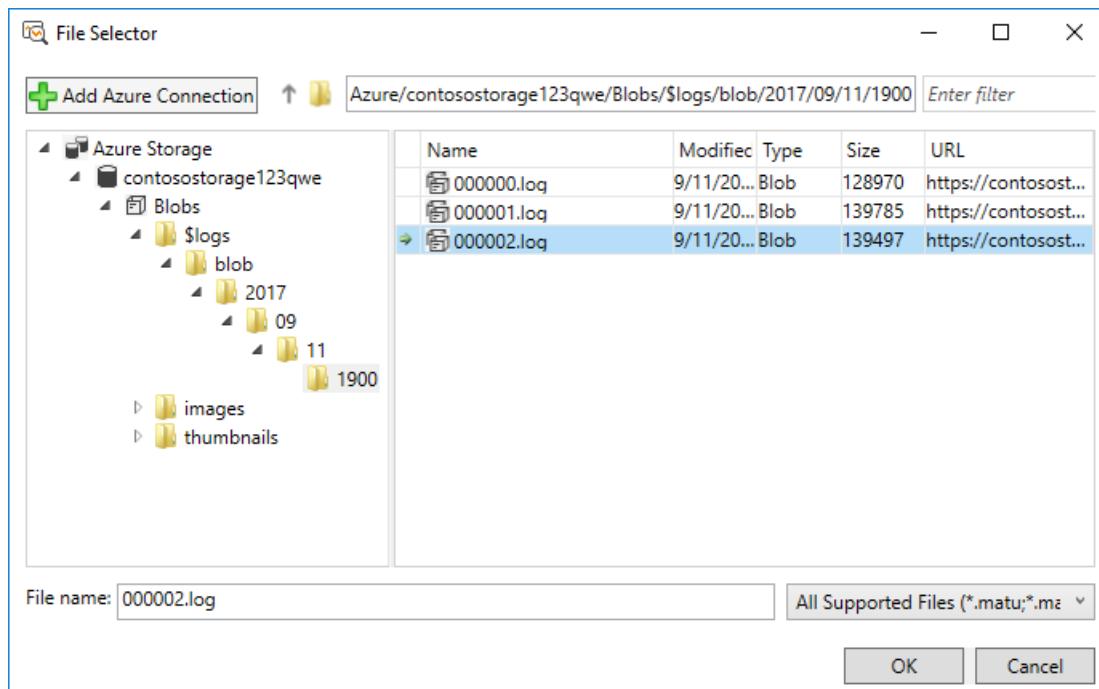
Download [Microsoft Message Analyzer](#) and install the application.

Launch the application and choose **File > Open > From Other File Sources**.

In the File Selector dialog, select + Add Azure Connection. Enter in your storage account name and account key and click OK.



Once you are connected, expand the containers in the storage tree view to view the log blobs. Select the latest log and click OK.



On the **New Session** dialog, click **Start** to view your log.

Once the log opens, you can view the storage events. As you can see from the following image, there was an `SASClientOtherError` triggered on the storage account. For additional information on storage logging, visit [Storage Analytics](#).

The screenshot shows the Microsoft Message Analyzer interface. The main pane displays a list of messages from a session. A red box highlights message 127, which has a green hexagonal icon. The message details are shown in the 'Details 1' pane:

Name	Value
Text	1.0;2017-09-08T21:34:41.5438330Z;GetBlob;SASClientOtherError;404;6;sas;;contosostorage123qwe;blob;"https://contosostorage123qwe.blob.core.windows.net/test?sv=2017-09-08&sr=b&sig=5438330Z&st=2017-09-08T21:34:41Z&se=2017-09-08T21:34:41Z&sp=r";

The 'Field Chooser' pane on the right lists various field categories. The 'Message Data 1' pane at the bottom right shows the selected message data.

[Storage Explorer](#) is another tool that can be used to interact with your storage accounts, including the \$logs container and the logs that are contained in it.

## Next steps

In part four and the final part of the series, you learned how to monitor and troubleshoot your storage account, such as how to:

- Turn on logging and metrics
  - Enable alerts for authorization errors
  - Run test traffic with incorrect SAS tokens
  - Download and analyze logs

Follow this link to see pre-built storage samples.

## Azure storage script samples

# Tutorial: Migrate on-premises data to cloud storage with AzCopy

11/21/2019 • 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.

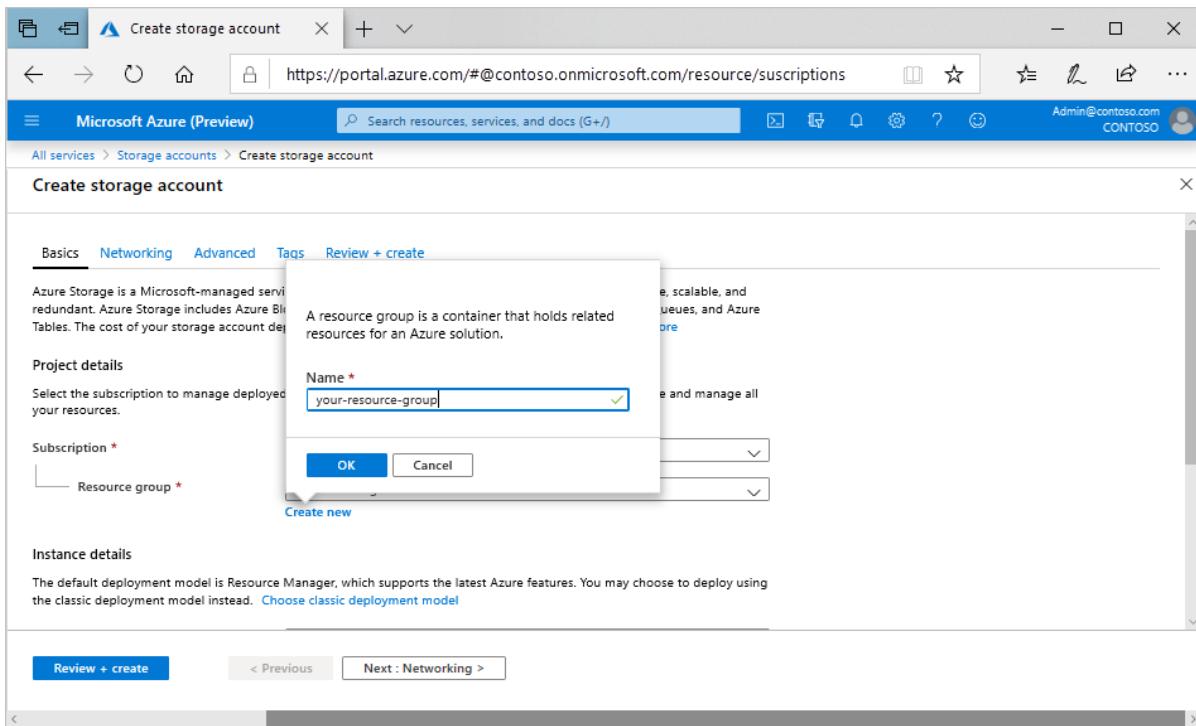
## Prerequisites

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. On the Azure portal menu, select **All services**. In the list of resources, type **Storage Accounts**. As you begin typing, the list filters based on your input. Select **Storage Accounts**.
2. On the **Storage Accounts** window that appears, choose **Add**.
3. Select the subscription in which to create the storage account.
4. Under the **Resource group** field, select **Create new**. Enter a name for your new resource group, as shown in the following image.



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 can include numbers and lowercase letters only.
6. Select a location for your storage account, or use the default location.
7. Leave these fields set to their default values:

FIELD	VALUE
Deployment model	Resource Manager
Performance	Standard
Account kind	StorageV2 (general-purpose v2)
Replication	Read-access geo-redundant storage (RA-GRS)
Access tier	Hot

8. If you plan to use [Azure Data Lake Storage](#), choose the **Advanced** tab, and then set **Hierarchical namespace** to **Enabled**.
9. Select **Review + Create** to review your storage account settings and create the account.
10. Select **Create**.

For more information about types of storage accounts and other storage account settings, see [Azure storage account overview](#). For more information on resource groups, see [Azure Resource Manager overview](#).

## 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. On the left, the navigation menu is open, showing various services like Storage accounts, App Services, and SQL databases. Under Storage accounts, there is a list of storage accounts, with one named 'ragrs' selected. The main content area shows the 'ragrs - Blobs' storage account details. A modal window titled 'New container' is open, prompting for a 'Name' (which is empty) and a 'Public access level' (set to 'Private (no anonymous access)'). Below the modal, there are sections for 'Settings' (Access keys, CORS, Configuration, Encryption, Shared access signature, Firewalls and virtual networks, Advanced Threat Protection, Static website, Properties, Locks, Automation script) and 'Blob service' (Blobs, Custom domain, Soft delete, Azure CDN, Add Azure Search).

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 [Grant access to Azure blob and queue data with RBAC in the Azure portal](#).

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.



## Enter code

Enter the code displayed on your app or device.

Code

Next

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%2BE2JHGCTRv52445rtoyqFBUo%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
- Configure, optimize, and troubleshoot AzCopy

# Create a virtual machine and storage account for a scalable application

12/10/2019 • 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 has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

## 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 <a href="#">Azure portal</a> .	

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

12/23/2019 • 6 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

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 application sets the minimum threads to 100 and the [DefaultConnectionLimit](#) to 100 to ensure that a large number of concurrent connections are allowed when running the application.

In addition to setting the threading and connection limit settings, the [BlobRequestOptions](#) for the [UploadFromStreamAsync](#) method are configured to use parallelism and disable MD5 hash validation. The files are uploaded in 100-mb blocks, this configuration provides better performance but can be costly if using a poorly performing network as if there is a failure the entire 100-mb block is retried.

PROPERTY	VALUE	DESCRIPTION
<a href="#">ParallelOperationThreadCount</a>	8	The setting breaks the blob into blocks when uploading. For highest performance, this value should be eight times the number of cores.
<a href="#">DisableContentMD5Validation</a>	true	This property disables checking the MD5 hash of the content uploaded. Disabling MD5 validation produces a faster transfer. But does not confirm the validity or integrity of the files being transferred.
<a href="#">StoreBlobContentMD5</a>	false	This property determines if an MD5 hash is calculated and stored with the file.
<a href="#">RetryPolicy</a>	2-second backoff with 10 max retry	Determines the retry policy of requests. Connection failures are retried, in this example an <a href="#">ExponentialRetry</a> policy is configured with a 2-second backoff, and a maximum retry count of 10. This setting is important when your application gets close to hitting the scalability targets for Blob storage. For more information, see <a href="#">Scalability and performance targets for Blob storage</a> .

The `UploadFilesAsync` task is shown in the following example:

```
private static async Task UploadFilesAsync()
{
 // Create random 5 characters containers to upload files to.
 CloudBlobContainer[] containers = await GetRandomContainersAsync();
 var currentdir = System.IO.Directory.GetCurrentDirectory();
```

```

// path to the directory to upload
string uploadPath = currentdir + "\\upload";
Stopwatch time = Stopwatch.StartNew();
try
{
 Console.WriteLine("Iterating in directory: {0}", uploadPath);
 int count = 0;
 int max_outstanding = 100;
 int completed_count = 0;

 // Define the BlobRequestOptions on the upload.
 // This includes defining an exponential retry policy to ensure that failed connections are retried
 // with a backoff policy. As multiple large files are being uploaded
 // large block sizes this can cause an issue if an exponential retry policy is not defined.
 // Additionally parallel operations are enabled with a thread count of 8
 // This could be should be multiple of the number of cores that the machine has. Lastly MD5 hash
 // validation is disabled for this example, this improves the upload speed.
 BlobRequestOptions options = new BlobRequestOptions
 {
 ParallelOperationThreadCount = 8,
 DisableContentMD5Validation = true,
 StoreBlobContentMD5 = false
 };
 // Create a new instance of the SemaphoreSlim class to define the number of threads to use in the
 // application.
 SemaphoreSlim sem = new SemaphoreSlim(max_outstanding, max_outstanding);

 List<Task> tasks = new List<Task>();
 Console.WriteLine("Found {0} file(s)", Directory.GetFiles(uploadPath).Count());

 // Iterate through the files
 foreach (string path in Directory.GetFiles(uploadPath))
 {
 // Create random file names and set the block size that is used for the upload.
 var container = containers[count % 5];
 string fileName = Path.GetFileName(path);
 Console.WriteLine("Uploading {0} to container {1}.", path, container.Name);
 CloudBlockBlob blockBlob = container.GetBlockBlobReference(fileName);

 // Set block size to 100MB.
 blockBlob.StreamWriteSizeInBytes = 100 * 1024 * 1024;
 await sem.WaitAsync();

 // Create tasks for each file that is uploaded. This is added to a collection that executes them
 // all asynchronously.
 tasks.Add(blockBlob.UploadFromFileAsync(path, null, options, null).ContinueWith((t) =>
 {
 sem.Release();
 Interlocked.Increment(ref completed_count);
 }));
 count++;
 }

 // Creates an asynchronous task that completes when all the uploads complete.
 await Task.WhenAll(tasks);

 time.Stop();

 Console.WriteLine("Upload has been completed in {0} seconds. Press any key to continue",
time.Elapsed.TotalSeconds.ToString());

 Console.ReadLine();
}
catch (DirectoryNotFoundException ex)
{
 Console.WriteLine("Error parsing files in the directory: {0}", ex.Message);
}
catch (Exception ex)

```

```

 catch (Exception ex)
 {
 Console.WriteLine(ex.Message);
 }
}

```

The following example is a truncated application output running on a Windows system.

```

Created container https://mystorageaccount.blob.core.windows.net/9efa7ecb-2b24-49ff-8e5b-1d25e5481076
Created container https://mystorageaccount.blob.core.windows.net/bbe5f0c8-be9e-4fc3-bcbd-2092433dbf6b
Created container https://mystorageaccount.blob.core.windows.net/9ac2f71c-6b44-40e7-b7be-8519d3ba4e8f
Created container https://mystorageaccount.blob.core.windows.net/47646f1a-c498-40cd-9dae-840f46072180
Created container https://mystorageaccount.blob.core.windows.net/38b2cdab-45fa-4cf9-94e7-d533837365aa
Iterating in directory: D:\git\storage-dotnet-perf-scale-app\upload
Found 50 file(s)
Starting upload of D:\git\storage-dotnet-perf-scale-app\upload\1d596d16-f6de-4c4c-8058-50ebd8141e4d.txt to
container 9efa7ecb-2b24-49ff-8e5b-1d25e5481076.
Starting upload of D:\git\storage-dotnet-perf-scale-app\upload\242ff392-78be-41fb-b9d4-aee8152a6279.txt to
container bbe5f0c8-be9e-4fc3-bcbd-2092433dbf6b.
Starting upload of D:\git\storage-dotnet-perf-scale-app\upload\38d4d7e2-acb4-4efc-ba39-f9611d0d55ef.txt to
container 9ac2f71c-6b44-40e7-b7be-8519d3ba4e8f.
Starting upload of D:\git\storage-dotnet-perf-scale-app\upload\45930d63-b0d0-425f-a766-cda27ff00d32.txt to
container 47646f1a-c498-40cd-9dae-840f46072180.
Starting upload of D:\git\storage-dotnet-perf-scale-app\upload\5129b385-5781-43be-8bac-e2fbb7d2bd82.txt to
container 38b2cdab-45fa-4cf9-94e7-d533837365aa.
...
Upload has been completed in 142.0429536 seconds. Press any key to continue

```

## Validate the connections

While the files are being uploaded, you can verify the number of concurrent connections to your storage account. Open a **Command Prompt** and type `netstat -a | find /c "blob:https"`. This command shows the number of connections that are currently opened using `netstat`. The following example shows a similar output to what you see when running the tutorial yourself. As you can see from the 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

12/10/2019 • 4 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.
 UploadFilesAsync().GetAwaiter().GetResult();

 // 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.
 // DownloadFilesAsync().GetAwaiter().GetResult();
 }
 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)
 {
 // DeleteExistingContainersAsync().GetAwaiter().GetResult();
 }
 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 application reads the containers located in the storage account specified in the `storageconnectionstring`. It iterates through the blobs 10 at a time using the `ListBlobsSegmented` method in the containers and downloads them to the local machine using the `DownloadToFileAsync` method. The following table shows the

[BlobRequestOptions](#) that are defined for each blob as it is downloaded.

PROPERTY	VALUE	DESCRIPTION
DisableContentMD5Validation	true	This property disables checking the MD5 hash of the content uploaded. Disabling MD5 validation produces a faster transfer. But does not confirm the validity or integrity of the files being transferred.
StoreBlobContentMD5	false	This property determines if an MD5 hash is calculated and stored.

The `DownloadFilesAsync` task is shown in the following example:

```
private static async Task DownloadFilesAsync()
{
 CloudBlobClient blobClient = GetCloudBlobClient();

 // Define the BlobRequestOptions on the download, including disabling MD5 hash validation for this
 // example, this improves the download speed.
 BlobRequestOptions options = new BlobRequestOptions
 {
 DisableContentMD5Validation = true,
 StoreBlobContentMD5 = false
 };

 // Retrieve the list of containers in the storage account. Create a directory and configure variables for
 // use later.
 BlobContinuationToken continuationToken = null;
 List<CloudBlobContainer> containers = new List<CloudBlobContainer>();
 do
 {
 var listingResult = await blobClient.ListContainersSegmentedAsync(continuationToken);
 continuationToken = listingResult.ContinuationToken;
 containers.AddRange(listingResult.Results);
 }
 while (continuationToken != null);

 var directory = Directory.CreateDirectory("download");
 BlobResultSegment resultSegment = null;
 Stopwatch time = Stopwatch.StartNew();

 // Download the blobs
 try
 {
 List<Task> tasks = new List<Task>();
 int max_outstanding = 100;
 int completed_count = 0;

 // Create a new instance of the SemaphoreSlim class to define the number of threads to use in the
 // application.
 SemaphoreSlim sem = new SemaphoreSlim(max_outstanding, max_outstanding);

 // Iterate through the containers
 foreach (CloudBlobContainer container in containers)
 {
 do
 {
 // Return the blobs from the container lazily 10 at a time.
 resultSegment = await container.ListBlobsSegmentedAsync(null, true, BlobListingDetails.All,
 10, continuationToken, null, null);
 continuationToken = resultSegment.ContinuationToken;
 }
 while (continuationToken != null);
 tasks.Add(resultSegment.DownloadToDirectory(directory));
 }
 await Task.WhenAll(tasks);
 }
 catch (Exception ex)
 {
 Console.WriteLine(ex.Message);
 }
}
```

```

 foreach (var blobItem in resultSegment.Results)
 {

 if (((CloudBlob)blobItem).Properties.BlobType == BlobType.BlockBlob)
 {
 // Get the blob and add a task to download the blob asynchronously from the
 storage account.
 CloudBlockBlob blockBlob =
 container.GetBlockBlobReference(((CloudBlockBlob)blobItem).Name);
 Console.WriteLine("Downloading {0} from container {1}", blockBlob.Name,
 container.Name);
 await sem.WaitAsync();
 tasks.Add(blockBlob.DownloadToFileAsync(directory.FullName + "\\\" +
 blockBlob.Name, FileMode.Create, null, options, null).ContinueWith((t) =>
 {
 sem.Release();
 Interlocked.Increment(ref completed_count);
 }));
 }
 }
 }
 while (continuationToken != null);
}

// Creates an asynchronous task that completes when all the downloads complete.
await Task.WhenAll(tasks);
}
catch (Exception e)
{
 Console.WriteLine("\nError encountered during transfer: {0}", e.Message);
}

time.Stop();
Console.WriteLine("Download has been completed in {0} seconds. Press any key to continue",
time.Elapsed.TotalSeconds.ToString());
Console.ReadLine();
}

```

## Validate the connections

While the files are being downloaded, you can verify the number of concurrent connections to your storage account. Open a `Command Prompt` and type `netstat -a | find /c "blob:https"`. This command shows the number of connections that are currently opened using `netstat`. The following example shows a similar output to what you see when running the tutorial yourself. As you can see from the example, over 280 connections were open when downloading the random 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 random data from a storage account, such as how to:

- Run the application
- Validate the number of connections

Advance 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

7/31/2019 • 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 (preview)** 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.

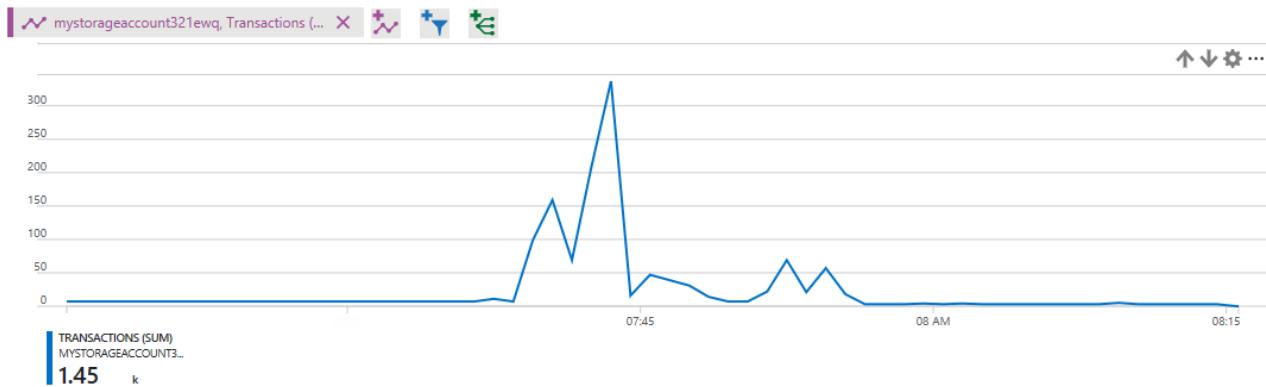
Select Last 24 hours (Automatic) next to Time. Choose Last hour and Minute for Time granularity, then click Apply.

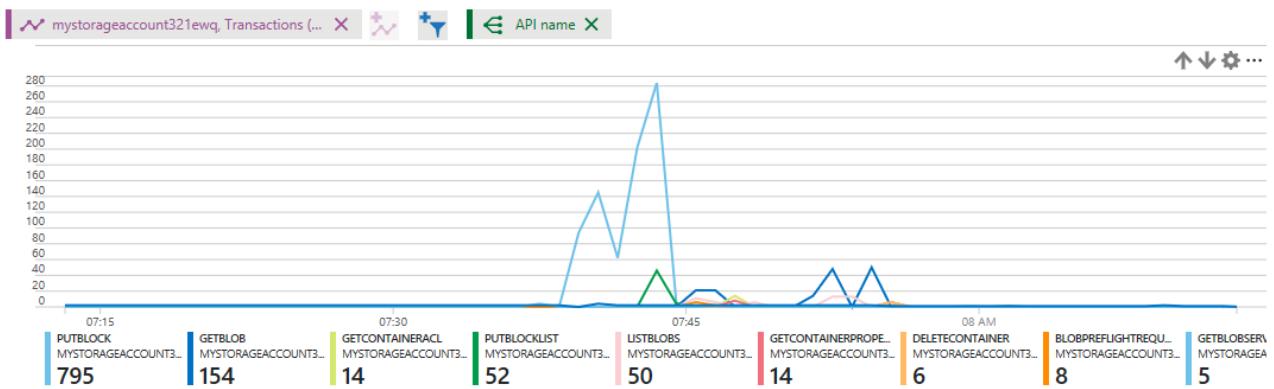
The screenshot shows the 'Metrics (preview)' blade for a storage account named 'mystorageaccount321ewq'. On the left, the 'Metrics (preview)' option is highlighted. In the center, a 'Time' configuration dialog is open, with 'Last hour' selected under 'Time granularity'. The main pane displays a chart for 'Egress' metric, with the Y-axis ranging from 0 to 100 and the X-axis showing times from 07:30 to 08:15. The chart shows a single data series for 'Egress'.

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

3/5/2020 • 3 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

## 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](#).

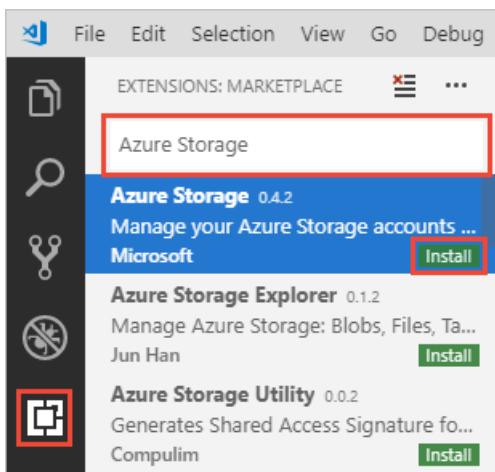
### NOTE

Make sure to create a general-purpose v2 Standard storage account . Static websites aren't available in any other type of storage account.

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.

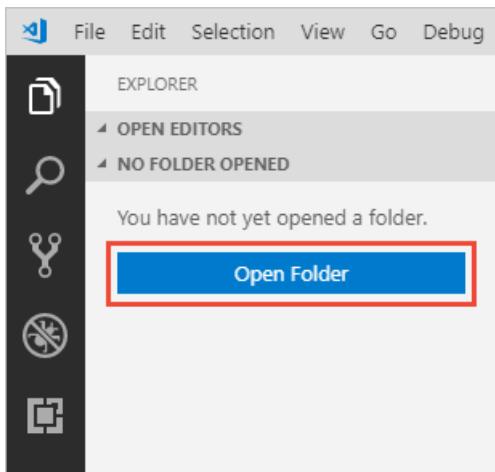
1. Open the [Azure portal](#) in your web browser.
2. Locate your storage account and display the account overview.
3. Select **Static website** to display the configuration page for static websites.
4. Select **Enabled** to enable static website hosting for the storage account.
5. In the **Index document 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.

The screenshot shows the Azure portal interface for configuring a static website. The top navigation bar includes 'Microsoft Azure (Preview)', a search bar, and a 'Save' button. The main area shows the 'Contoso' storage account under 'Static website'. A sidebar on the left lists various configuration options: Access keys, Geo-replication, CORS, Configuration, Encryption, Shared access signature, Firewalls and virtual networks, Private endpoint connection..., Advanced security, Static website (which is selected and highlighted in grey), and Properties. The main content area contains a note about enabling static websites on the blob service, stating it allows hosting static content like webpages and client-side scripts. It shows the 'Static website' section where 'Enabled' is selected. Below this, it indicates that an Azure Storage container has been created for the static website, specifically '\$web'. The 'Primary endpoint' is listed as `https://contoso.web.core.windows.net/`. The 'Index document name' is set to `index.html`, and the 'Error document path' is set to `404.html`.

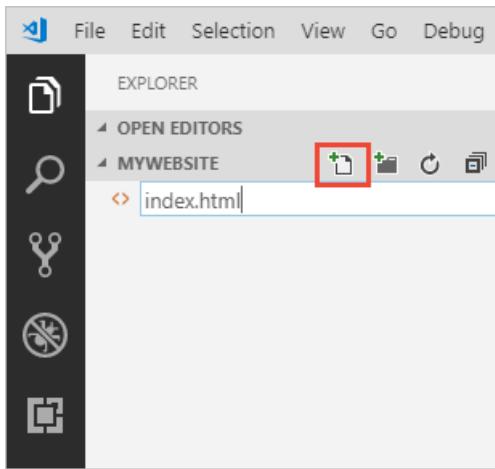
## Deploy a Hello World website

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:

```
<h1>Hello World!</h1>
```

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:

```
<h1>404</h1>
```

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.

## 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

4/17/2020 • 10 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-redundant \(RA-GRS\) storage account](#).

RA-GRS works by replicating transactions from a primary region to a secondary region. This replication process guarantees that the data in the secondary region is eventually consistent. The 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.

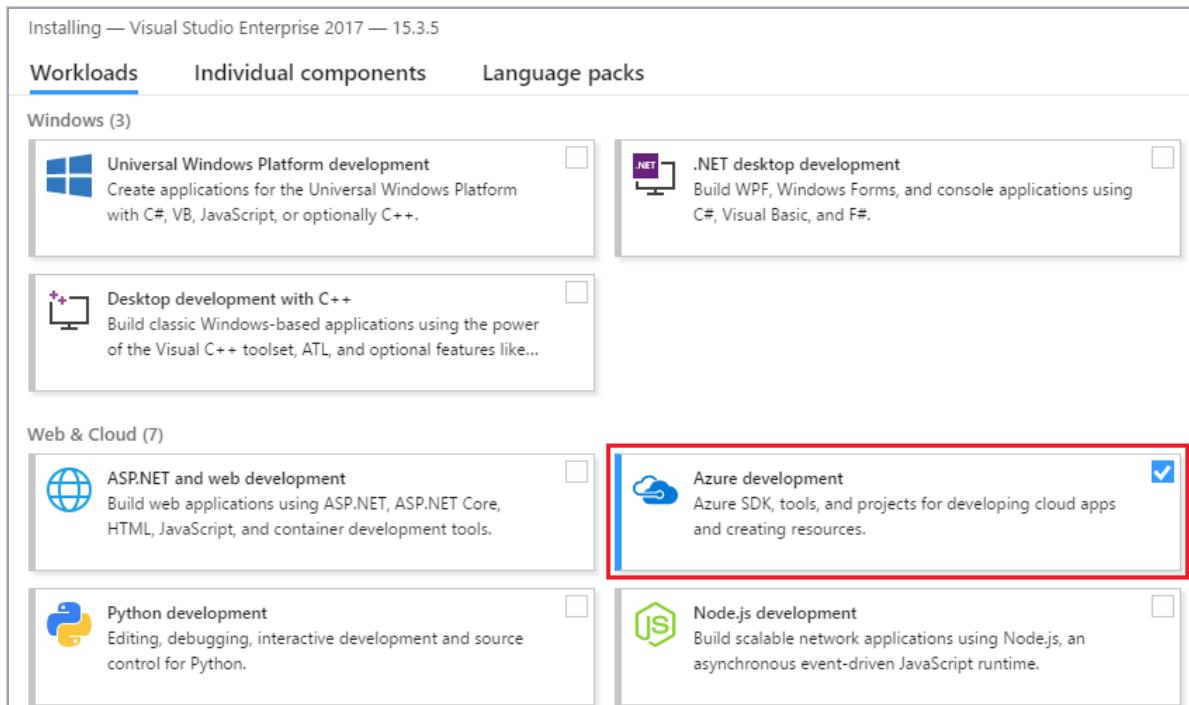
In part one of the series, you learn how to:

- Create a storage account
- Set the connection string
- Run the console application

## Prerequisites

To complete this tutorial:

- [.NET](#)
- [Python](#)
- [Node.js](#)
- Install [Visual Studio 2019](#) with the [Azure development](#) workload.



# Sign in to the Azure portal

Sign in to the [Azure portal](#).

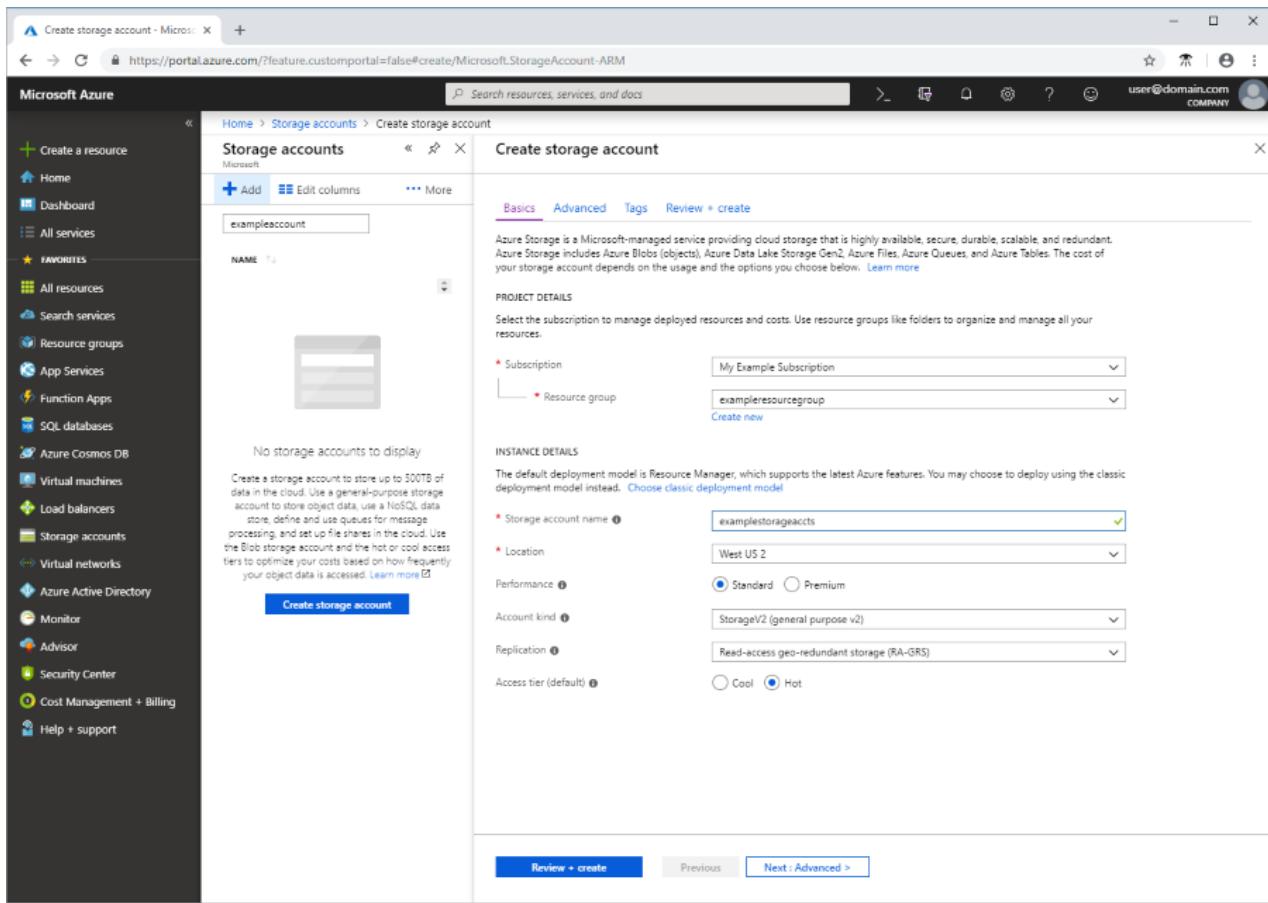
## Create a storage account

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-redundant storage account:

1. Select the **Create a resource** button found on the upper left-hand corner of the Azure portal.
2. Select **Storage** from the **New** page.
3. Select **Storage account - blob, file, table, queue** under **Featured**.
4. Fill out the storage account form with the following information, as shown in the following image and select **Create**:

SETTING	SUGGESTED VALUE	DESCRIPTION
Name	mystorageaccount	A unique value for your storage account
Deployment model	Resource Manager	Resource Manager contains the latest features.
Account kind	StorageV2	For details on the types of accounts, see <a href="#">types of storage accounts</a>
Performance	Standard	Standard is sufficient for the example scenario.
Replication	Read-access geo-redundant storage (RA-GRS)	This is necessary for the sample to work.
Subscription	your subscription	For details about your subscriptions, see <a href="#">Subscriptions</a> .
ResourceGroup	myResourceGroup	For valid resource group names, see <a href="#">Naming rules and restrictions</a> .
Location	East US	Choose a location.



## Download the sample

- [.NET](#)
- [Python](#)
- [Node.js](#)

[Download the sample project](#) and extract (unzip) the storage-dotnet-circuit-breaker-pattern-ha-apps-using-ra-grs.zip file. You can also use [git](#) to download a copy of the application to your development environment. The sample project contains a console application.

```
git clone https://github.com/Azure-Samples/storage-dotnet-circuit-breaker-pattern-ha-apps-using-ra-grs.git
```

## Configure the sample

- [.NET](#)
- [Python](#)
- [Node.js](#)

In the application, you must provide the connection string for your storage account. You can store this connection string within an environment variable on the local machine running the application. Follow one of the examples below depending on your Operating System to create the environment variable.

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. Run one of the following commands based on your operating system, replacing <yourconnectionstring> with your actual connection string. This command saves an environment variable to the local machine. In Windows, the environment variable is not available until you reload the **Command Prompt** or shell you are using.

### Linux

```
export storageconnectionstring=<yourconnectionstring>
```

## Windows

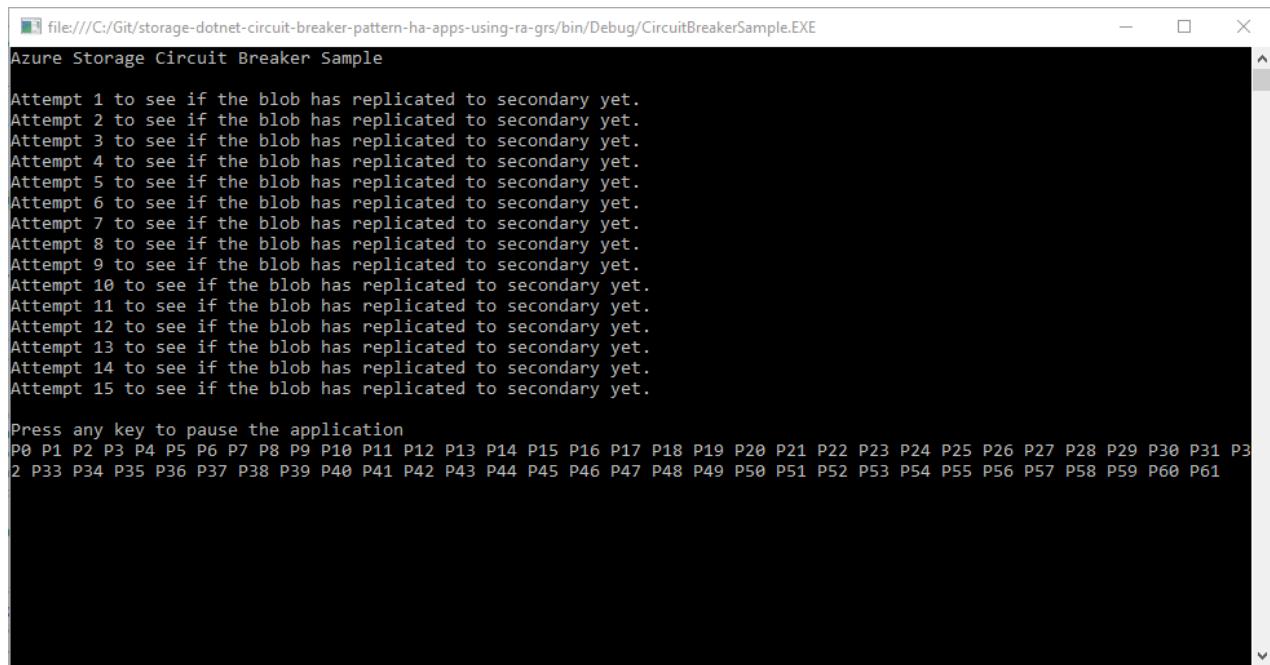
```
setx storageconnectionstring "<yourconnectionstring>"
```

## Run the console application

- [.NET](#)
- [Python](#)
- [Node.js](#)

In Visual Studio, press **F5** or select **Start** to begin debugging the application. Visual studio automatically restores missing NuGet packages if configured, visit [Installing and reinstalling packages with package restore](#) to learn more.

A console window launches and the application begins running. The application uploads the **HelloWorld.png** image from the solution to the storage account. The application checks to ensure the image has replicated to the secondary RA-GRS endpoint. It then begins downloading the image up to 999 times. Each read is represented by a P or an S. Where P represents the primary endpoint and S represents the secondary endpoint.



```
file:///C:/Git/storage-dotnet-circuit-breaker-pattern-ha-apps-using-ra-grs/bin/Debug/CircuitBreakerSample.EXE
Azure Storage Circuit Breaker Sample

Attempt 1 to see if the blob has replicated to secondary yet.
Attempt 2 to see if the blob has replicated to secondary yet.
Attempt 3 to see if the blob has replicated to secondary yet.
Attempt 4 to see if the blob has replicated to secondary yet.
Attempt 5 to see if the blob has replicated to secondary yet.
Attempt 6 to see if the blob has replicated to secondary yet.
Attempt 7 to see if the blob has replicated to secondary yet.
Attempt 8 to see if the blob has replicated to secondary yet.
Attempt 9 to see if the blob has replicated to secondary yet.
Attempt 10 to see if the blob has replicated to secondary yet.
Attempt 11 to see if the blob has replicated to secondary yet.
Attempt 12 to see if the blob has replicated to secondary yet.
Attempt 13 to see if the blob has replicated to secondary yet.
Attempt 14 to see if the blob has replicated to secondary yet.
Attempt 15 to see if the blob has replicated to secondary yet.

Press any key to pause the application
P0 P1 P2 P3 P4 P5 P6 P7 P8 P9 P10 P11 P12 P13 P14 P15 P16 P17 P18 P19 P20 P21 P22 P23 P24 P25 P26 P27 P28 P29 P30 P31 P32 P33 P34 P35 P36 P37 P38 P39 P40 P41 P42 P43 P44 P45 P46 P47 P48 P49 P50 P51 P52 P53 P54 P55 P56 P57 P58 P59 P60 P61
```

In the sample code, the `RunCircuitBreakerAsync` task in the `Program.cs` file is used to download an image from the storage account using the `DownloadToFileAsync` method. Prior to the download, an `OperationContext` is defined. The operation context defines event handlers, that fire when a download completes successfully or if a download fails and is retrying.

## Understand the sample code

- [.NET](#)
- [Python](#)
- [Node.js](#)

### Retry event handler

The `OperationContextRetrying` event handler is called when the download of the image fails and is set to retry. If

the maximum number of retries defined in the application are reached, the [LocationMode](#) of the request is changed to `SecondaryOnly`. This setting forces the application to attempt to download the image from the secondary endpoint. This configuration reduces the time taken to request the image as the primary endpoint is not retried indefinitely.

```
private static void OperationContextRetrying(object sender, RequestEventArgs e)
{
 retryCount++;
 Console.WriteLine("Retrying event because of failure reading the primary. RetryCount = " + retryCount);

 // Check if we have had more than n retries in which case switch to secondary.
 if (retryCount >= retryThreshold)
 {

 // Check to see if we can fail over to secondary.
 if (blobClient.DefaultRequestOptions.LocationMode != LocationMode.SecondaryOnly)
 {
 blobClient.DefaultRequestOptions.LocationMode = LocationMode.SecondaryOnly;
 retryCount = 0;
 }
 else
 {
 throw new ApplicationException("Both primary and secondary are unreachable. Check your
application's network connection. ");
 }
 }
}
```

## Request completed event handler

The `OperationContextRequestCompleted` event handler is called when the download of the image is successful. If the application is using the secondary endpoint, the application continues to use this endpoint up to 20 times. After 20 times, the application sets the [LocationMode](#) back to `PrimaryThenSecondary` and retries the primary endpoint. If a request is successful, the application continues to read from the primary endpoint.

```
private static void OperationContextRequestCompleted(object sender, RequestEventArgs e)
{
 if (blobClient.DefaultRequestOptions.LocationMode == LocationMode.SecondaryOnly)
 {
 // You're reading the secondary. Let it read the secondary [secondaryThreshold] times,
 // then switch back to the primary and see if it's available now.
 secondaryReadCount++;
 if (secondaryReadCount >= secondaryThreshold)
 {
 blobClient.DefaultRequestOptions.LocationMode = LocationMode.PrimaryThenSecondary;
 secondaryReadCount = 0;
 }
 }
}
```

## Next steps

In part one of the series, you learned about making an application highly available with RA-GRS 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-GRS endpoint.

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

# Tutorial: Simulate a failure in reading data from the primary region

3/20/2020 • 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-redundant storage \(RA-GRS\)](#) 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-GRS) storage account, causing the application 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-GRS) 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.

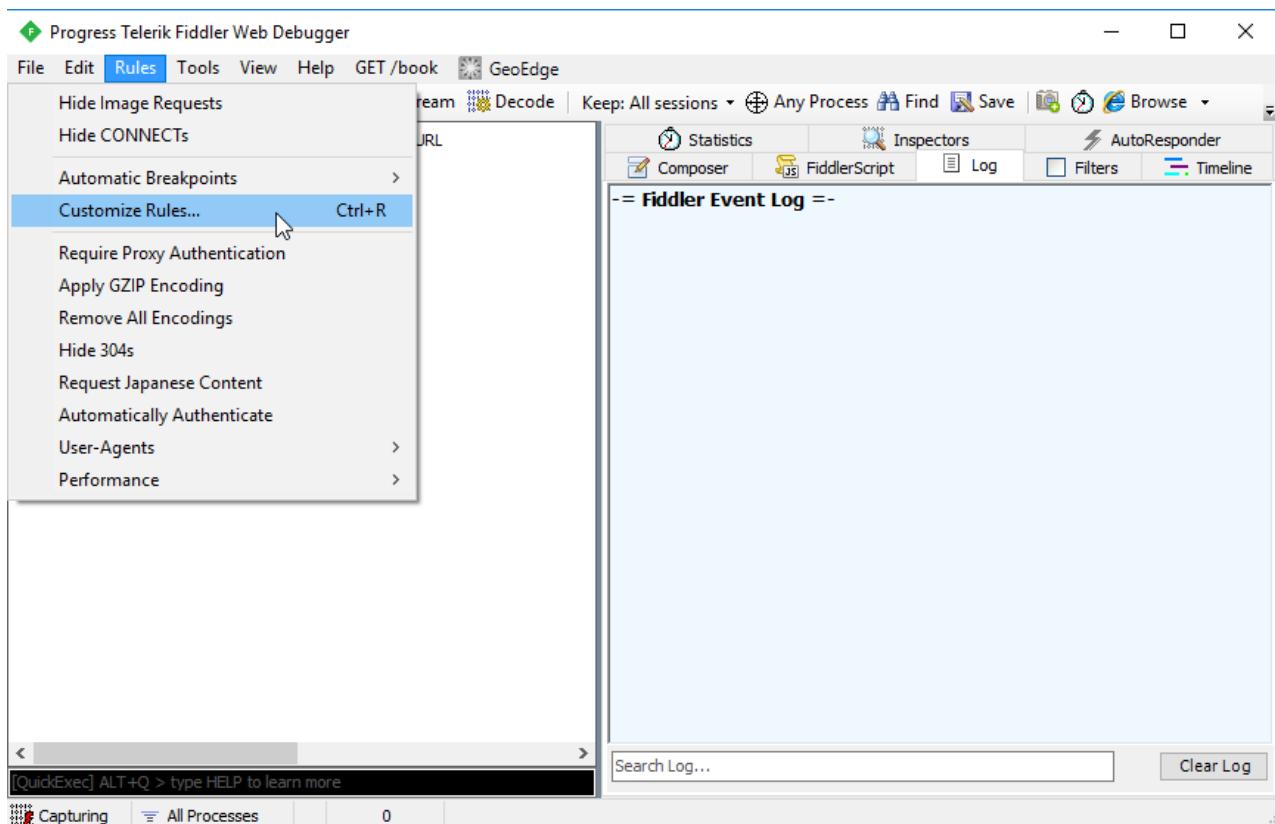
## Simulate a failure with Fiddler

To simulate failure with Fiddler, you inject a failed response for requests to the primary endpoint of your RA-GRS 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 syntax highlighting for JavaScript-like code. The right pane is a sidebar titled "Click on an item for information" containing a list of Fiddler objects and methods.

```
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;
 */
}
/*
<
```

287:39

FiddlerObject  
FiddlerApplication  
FiddlerApplication.UI  
FiddlerApplication.oProxy  
CONFIG  
Utilities  
Session  
HTTPRequestHeaders  
HTTPResponseHeaders  
WebSocketMessage

## 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-GRS 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-GRS storage works, as well as its associated risks, read the following article:

[Designing HA apps with RA-GRS](#)

# Tutorial - Encrypt and decrypt blobs using Azure Key Vault

4/16/2020 • 6 minutes to read • [Edit Online](#)

This tutorial covers how to make use of client-side storage encryption with Azure Key Vault. It walks you through how to encrypt and decrypt a blob in a console application using these technologies.

**Estimated time to complete:** 20 minutes

For overview information about Azure Key Vault, see [What is Azure Key Vault?](#).

For overview information about client-side encryption for Azure Storage, see [Client-Side Encryption and Azure Key Vault for Microsoft Azure Storage](#).

## Prerequisites

To complete this tutorial, you must have the following:

- An Azure Storage account
- Visual Studio 2013 or later
- Azure PowerShell

## Overview of client-side encryption

For an overview of client-side encryption for Azure Storage, see [Client-Side Encryption and Azure Key Vault for Microsoft Azure Storage](#)

Here is a brief description of how client side encryption works:

1. The Azure Storage client SDK generates a content encryption key (CEK), which is a one-time-use symmetric key.
2. Customer data is encrypted using this 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 an asymmetric key pair or a symmetric key and can be managed locally or stored in Azure Key Vault. The Storage client itself never has access to the KEK. It just invokes the key wrapping algorithm that is provided by Key Vault. Customers can choose to use custom providers for key wrapping/unwrapping if they want.
4. The encrypted data is then uploaded to the Azure Storage service.

## Set up your Azure Key Vault

In order to proceed with this tutorial, you need to do the following steps, which are outlined in the tutorial [Quickstart: Set and retrieve a secret from Azure Key Vault by using a .NET web app](#):

- Create a key vault.
- Add a key or secret to the key vault.
- Register an application with Azure Active Directory.
- Authorize the application to use the key or secret.

Make note of the ClientID and ClientSecret that were generated when registering an application with Azure Active Directory.

Create both keys in the key vault. We assume for the rest of the tutorial that you have used the following names:

ContosoKeyVault and TestRSAKey1.

## Create a console application with packages and AppSettings

In Visual Studio, create a new console application.

Add necessary nuget packages in the Package Manager Console.

```
Install-Package Microsoft.Azure.ConfigurationManager
Install-Package Microsoft.Azure.Storage.Common
Install-Package Microsoft.Azure.Storage.Blob
Install-Package Microsoft.IdentityModel.Clients.ActiveDirectory

Install-Package Microsoft.Azure.KeyVault
Install-Package Microsoft.Azure.KeyVault.Extensions
```

Add AppSettings to the App.Config.

```
<appSettings>
 <add key="accountName" value="myaccount"/>
 <add key="accountKey" value="theaccountkey"/>
 <add key="clientId" value="theclientid"/>
 <add key="clientSecret" value="theclientsecret"/>
 <add key="container" value="stuff"/>
</appSettings>
```

Add the following `using` directives and make sure to add a reference to System.Configuration to the project.

```
using Microsoft.IdentityModel.Clients.ActiveDirectory;
using System.Configuration;
using Microsoft.Azure;
using Microsoft.Azure.Storage;
using Microsoft.Azure.Storage.Auth;
using Microsoft.Azure.Storage.Blob;
using Microsoft.Azure.KeyVault;
using System.Threading;
using System.IO;
```

## Add a method to get a token to your console application

The following method is used by Key Vault classes that need to authenticate for access to your key vault.

```
private async static Task<string> GetToken(string authority, string resource, string scope)
{
 var authContext = new AuthenticationContext(authority);
 ClientCredential clientCred = new ClientCredential(
 CloudConfigurationManager.GetSetting("clientId"),
 CloudConfigurationManager.GetSetting("clientSecret"));
 AuthenticationResult result = await authContext.AcquireTokenAsync(resource, clientCred);

 if (result == null)
 throw new InvalidOperationException("Failed to obtain the JWT token");

 return result.AccessToken;
}
```

## Access Azure Storage and Key Vault in your program

In the Main() method, add the following code.

```
// This is standard code to interact with Blob storage.
StorageCredentials creds = new StorageCredentials(
 CloudConfigurationManager.GetSetting("accountName"),
 CloudConfigurationManager.GetSetting("accountKey"))
);
CloudStorageAccount account = new CloudStorageAccount(creds, useHttps: true);
CloudBlobClient client = account.CreateCloudBlobClient();
CloudBlobContainer contain = client.GetContainerReference(CloudConfigurationManager.GetSetting("container"));
contain.CreateIfNotExists();

// The Resolver object is used to interact with Key Vault for Azure Storage.
// This is where the GetToken method from above is used.
KeyVaultKeyResolver cloudResolver = new KeyVaultKeyResolver(GetToken);
```

#### NOTE

##### Key Vault Object Models

It is important to understand that there are actually two Key Vault object models to be aware of: one is based on the REST API (KeyVault namespace) and the other is an extension for client-side encryption.

The Key Vault Client interacts with the REST API and understands JSON Web Keys and secrets for the two kinds of things that are contained in Key Vault.

The Key Vault Extensions are classes that seem specifically created for client-side encryption in Azure Storage. They contain an interface for keys (IKey) and classes based on the concept of a Key Resolver. There are two implementations of IKey that you need to know: RSAKey and SymmetricKey. Now they happen to coincide with the things that are contained in a Key Vault, but at this point they are independent classes (so the Key and Secret retrieved by the Key Vault Client do not implement IKey).

## Encrypt blob and upload

Add the following code to encrypt a blob and upload it to your Azure storage account. The **ResolveKeyAsync** method that is used returns an IKey.

```
// Retrieve the key that you created previously.
// The IKey that is returned here is an RsaKey.
var rsa = cloudResolver.ResolveKeyAsync(
 "https://contosokeyvault.vault.azure.net/keys/TestRSAKey1",
 CancellationToken.None).GetAwaiter().GetResult();

// Now you simply use the RSA key to encrypt by setting it in the BlobEncryptionPolicy.
BlobEncryptionPolicy policy = new BlobEncryptionPolicy(rsa, null);
BlobRequestOptions options = new BlobRequestOptions() { EncryptionPolicy = policy };

// Reference a block blob.
CloudBlockBlob blob = contain.GetBlockBlobReference("MyFile.txt");

// Upload using the UploadFromStream method.
using (var stream = System.IO.File.OpenRead(@"C:\Temp\MyFile.txt"))
 blob.UploadFromStream(stream, stream.Length, null, options, null);
```

#### NOTE

If you look at the BlobEncryptionPolicy constructor, you will see that it can accept a key and/or a resolver. Be aware that right now you cannot use a resolver for encryption because it does not currently support a default key.

## Decrypt blob and download

Decryption is really when using the Resolver classes make sense. The ID of the key used for encryption is associated with the blob in its metadata, so there is no reason for you to retrieve the key and remember the association between key and blob. You just have to make sure that the key remains in Key Vault.

The private key of an RSA Key remains in Key Vault, so for decryption to occur, the Encrypted Key from the blob metadata that contains the CEK is sent to Key Vault for decryption.

Add the following to decrypt the blob that you just uploaded.

```
// In this case, we will not pass a key and only pass the resolver because
// this policy will only be used for downloading / decrypting.
BlobEncryptionPolicy policy = new BlobEncryptionPolicy(null, cloudResolver);
BlobRequestOptions options = new BlobRequestOptions() { EncryptionPolicy = policy };

using (var np = File.Open(@"C:\data\MyFileDecrypted.txt", FileMode.Create))
 blob.DownloadToStream(np, null, options, null);
```

### NOTE

There are a couple of other kinds of resolvers to make key management easier, including: AggregateKeyResolver and CachingKeyResolver.

## Use Key Vault secrets

The way to use a secret with client-side encryption is via the SymmetricKey class because a secret is essentially a symmetric key. But, as noted above, a secret in Key Vault does not map exactly to a SymmetricKey. There are a few things to understand:

- The key in a SymmetricKey has to be a fixed length: 128, 192, 256, 384, or 512 bits.
- The key in a SymmetricKey should be Base64 encoded.
- A Key Vault secret that will be used as a SymmetricKey needs to have a Content Type of "application/octet-stream" in Key Vault.

Here is an example in PowerShell of creating a secret in Key Vault that can be used as a SymmetricKey. Please note that the hard coded value, \$key, is for demonstration purpose only. In your own code you'll want to generate this key.

```
// Here we are making a 128-bit key so we have 16 characters.
// The characters are in the ASCII range of UTF8 so they are
// each 1 byte. 16 x 8 = 128.
$key = "qwertyuiopasdfgh"
$b = [System.Text.Encoding]::UTF8.GetBytes($key)
$enc = [System.Convert]::ToBase64String($b)
$secretvalue = ConvertTo-SecureString $enc -AsPlainText -Force

// Substitute the VaultName and Name in this command.
$secret = Set-AzureKeyVaultSecret -VaultName 'ContosoKeyVault' -Name 'TestSecret2' -SecretValue $secretvalue -
ContentType "application/octet-stream"
```

In your console application, you can use the same call as before to retrieve this secret as a SymmetricKey.

```
SymmetricKey sec = (SymmetricKey) cloudResolver.ResolveKeyAsync(
 "https://contosokeyvault.vault.azure.net/secrets/TestSecret2/",
 CancellationToken.None).GetAwaiter().GetResult();
```

That's it. Enjoy!

## Next steps

For more information about using Microsoft Azure Storage with C#, see [Microsoft Azure Storage Client Library for .NET](#).

For more information about the Blob REST API, see [Blob Service REST API](#).

For the latest information on Microsoft Azure Storage, go to the [Microsoft Azure Storage Team Blog](#).

# Azure Storage samples using v12 .NET client libraries

4/9/2020 • 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](#)
- All other languages: [Azure Storage samples](#)

# Azure Storage samples using v12 Java client libraries

2/20/2020 • 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](#)

[Create a file using a Data Lake file client](#)

[Append data to a file](#)

[Download 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](#)
- All other languages: [Azure Storage samples](#)

# Azure Storage samples using v12 Python client libraries

2/20/2020 • 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](#)
- All other languages: [Azure Storage samples](#)



# Azure Storage samples using v12 JavaScript client libraries

2/20/2020 • 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](#)
- All other languages: [Azure Storage samples](#)

# Azure PowerShell samples for Azure Blob storage

8/1/2019 • 2 minutes to read • [Edit Online](#)

The following table includes links to PowerShell script samples that create and manage Azure Storage.

<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

8/1/2019 • 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.

Storage accounts	
<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.
Blob storage	
<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.

# Storage account overview

3/13/2020 • 14 minutes to read • [Edit Online](#)

An Azure storage account contains all of your Azure Storage data objects: blobs, files, queues, tables, and disks. The storage account provides a unique namespace for your Azure Storage data that is accessible from anywhere in the world over HTTP or HTTPS. Data in your Azure 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. Consider these differences before you create a storage account to determine the type of account that is best for your applications. The types of storage accounts are:

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

The following table describes the types of storage accounts and their capabilities:

STORAGE ACCOUNT TYPE	SUPPORTED SERVICES	SUPPORTED PERFORMANCE TIERS	SUPPORTED ACCESS TIERS	REPLICATION OPTIONS	DEPLOYMENT MODEL 1	ENCRYPTION 2
General-purpose V2	Blob, File, Queue, Table, Disk, and Data Lake Gen2 <sup>6</sup>	Standard, Premium <sup>5</sup>	Hot, Cool, Archive <sup>3</sup>	LRS, GRS, RA-GRS, ZRS, GZRS (preview), RA-GZRS (preview) <sup>4</sup>	Resource Manager	Encrypted
General-purpose V1	Blob, File, Queue, Table, and Disk	Standard, Premium <sup>5</sup>	N/A	LRS, GRS, RA-GRS	Resource Manager, Classic	Encrypted
BlockBlobStorage	Blob (block blobs and append blobs only)	Premium	N/A	LRS, ZRS <sup>4</sup>	Resource Manager	Encrypted

Storage Account Type	Supported Services	Supported Performance Tiers	Supported Access Tiers	Replication Options	Deployment Model	Encryption
FileStorage	File only	Premium	N/A	LRS, ZRS <sup>4</sup>	Resource Manager	Encrypted
BlobStorage	Blob (block blobs and append blobs only)	Standard	Hot, Cool, Archive <sup>3</sup>	LRS, GRS, RA-GRS	Resource Manager	Encrypted

<sup>1</sup>Using the Azure Resource Manager deployment model is recommended. Storage accounts using the classic deployment model can still be created in some locations, and existing classic accounts continue to be supported. For more information, see [Azure Resource Manager vs. classic deployment: Understand deployment models and the state of your resources](#).

<sup>2</sup>All storage accounts are encrypted using Storage Service Encryption (SSE) for data at rest. For more information, see [Azure Storage Service Encryption for Data at Rest](#).

<sup>3</sup>Archive storage and blob-level tiering only support block blobs. The Archive tier is available at the level of an individual blob only, not at the storage account level. For more information, see [Azure Blob storage: Hot, Cool, and Archive storage tiers](#).

<sup>4</sup>Zone-redundant storage (ZRS) and geo-zone-redundant storage (GZRS/RA-GZRS) (preview) are available only for standard general-purpose V2, BlockBlobStorage, and FileStorage accounts in certain regions. For more information about Azure Storage redundancy options, see [Azure Storage redundancy](#).

<sup>5</sup>Premium performance for general-purpose v2 and general-purpose v1 accounts is available for disk and page blob only. Premium performance for block or append blobs are only available on BlockBlobStorage accounts. Premium performance for files are only available on FileStorage accounts.

<sup>6</sup>Azure Data Lake Storage Gen2 is a set of capabilities dedicated to big data analytics, built on Azure Blob storage. Data Lake Storage Gen2 is only supported on General-purpose V2 storage accounts with Hierarchical namespace enabled. For more information on Data Lake Storage Gen2, see [Introduction to Azure Data Lake Storage Gen2](#).

## General-purpose v2 accounts

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 deliver the lowest per-gigabyte capacity prices for Azure Storage, as well as industry-competitive transaction prices. General-purpose v2 storage accounts support these Azure Storage services:

- Blobs (all types: Block, Append, Page)
- Data Lake Gen2
- Files
- Disks
- Queues
- Tables

#### **NOTE**

Microsoft recommends using a general-purpose v2 storage account for most scenarios. You can easily upgrade a general-purpose v1 or Blob storage account to a general-purpose v2 account with no downtime and without the need to copy data.

For more information on upgrading to a general-purpose v2 account, see [Upgrade to a general-purpose v2 storage account](#).

General-purpose v2 storage accounts offer multiple access tiers for storing data based on your usage patterns. For more information, see [Access tiers for block blob data](#).

### **General-purpose v1 accounts**

General-purpose v1 storage accounts provide access to all Azure Storage services, but may not have the latest features or the lowest per gigabyte pricing. General-purpose v1 storage accounts support these Azure Storage services:

- Blobs (all types)
- Files
- Disks
- Queues
- Tables

You should use general-purpose v2 accounts in most cases. You can use general-purpose v1 accounts for these scenarios:

- Your applications require the Azure classic deployment model. General-purpose v2 accounts and Blob storage accounts support only the Azure Resource Manager deployment model.
- Your applications are transaction-intensive or use significant geo-replication bandwidth, but don't require large capacity. In this case, general-purpose v1 may be the most economical choice.
- You use a version of the [Storage Services REST API](#) that is earlier than 2014-02-14 or a client library with a version lower than 4.x. You can't upgrade your application.

### **BlockBlobStorage accounts**

A BlockBlobStorage account is a specialized storage account in the premium performance tier for storing unstructured object data as block blobs or append blobs. Compared with general-purpose v2 and BlobStorage accounts, BlockBlobStorage accounts provide low, consistent latency and higher transaction rates.

BlockBlobStorage accounts don't currently support tiering to hot, cool, or archive access tiers. This type of storage account does not support page blobs, tables, or queues.

### **FileStorage accounts**

A FileStorage account is a specialized storage account used to store and create premium file shares. This storage account kind supports files but not block blobs, append blobs, page blobs, tables, or queues.

FileStorage accounts offer unique performance dedicated characteristics such as IOPS bursting. For more information on these characteristics, see the [File share storage tiers](#) section of the Files planning guide.

## **Naming storage accounts**

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.

# Performance tiers

Depending on the type of storage account you create, you can choose between standard and premium performance tiers.

## General-purpose storage accounts

General-purpose storage accounts may be configured for either of the following performance tiers:

- A standard performance tier for storing blobs, files, tables, queues, and Azure virtual machine disks. For more information about scalability targets for standard storage accounts, see [Scalability targets for standard storage accounts](#).
- A premium performance tier for storing unmanaged virtual machine disks. Microsoft recommends using managed disks with Azure virtual machines instead of unmanaged disks. For more information about scalability targets for the premium performance tier, see [Scalability targets for premium page blob storage accounts](#).

## BlockBlobStorage storage accounts

BlockBlobStorage storage accounts provide a premium performance tier for storing block blobs and append blobs. For more information, see [Scalability targets for premium block blob storage accounts](#).

## FileStorage storage accounts

FileStorage storage accounts provide a premium performance tier for Azure file shares. For more information, see [Azure Files scalability and performance targets](#).

## Access tiers for block blob data

Azure Storage provides different options for accessing block blob data based on usage patterns. Each access tier in Azure Storage is optimized for a particular pattern of data usage. By selecting the right access tier for your needs, you can store your block blob data in the most cost-effective manner.

The available access tiers are:

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

If there's a change in the usage pattern of your data, you can switch between these access tiers at any time. For more information about access tiers, see [Azure Blob storage: hot, cool, and archive access tiers](#).

### IMPORTANT

Changing the access tier for an existing storage account or blob may result in additional charges. For more information, see the [Storage account billing section](#).

## Redundancy

Redundancy options for a storage account include:

- Locally redundant storage (LRS): A simple, low-cost redundancy strategy. Data is copied synchronously three

times within the primary region.

- Zone-redundant storage (ZRS): Redundancy for scenarios requiring high availability. Data is copied synchronously across three Azure availability zones in the primary region.
- Geo-redundant storage (GRS): Cross-regional redundancy to protect against regional outages. Data is copied synchronously three times in the primary region, then copied asynchronously to the secondary region. For read access to data in the secondary region, enable read-access geo-redundant storage (RA-GRS).
- Geo-zone-redundant storage (GZRS) (preview): Redundancy for scenarios requiring both high availability and maximum durability. Data is copied synchronously across three Azure availability zones in the primary region, then copied asynchronously to the secondary region. For read access to data in the secondary region, enable read-access geo-zone-redundant storage (RA-GZRS).

For more information about redundancy options in Azure Storage, see [Azure Storage redundancy](#).

## Encryption

All data in your storage account is encrypted on the service side. For more information about encryption, see [Azure Storage Service Encryption for data at rest](#).

## Storage account endpoints

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

For example, if your general-purpose storage account is named *mystorageaccount*, then the default endpoints for that account are:

- Blob storage: `https://*mystorageaccount*.blob.core.windows.net`
- Table storage: `https://*mystorageaccount*.table.core.windows.net`
- Queue storage: `https://*mystorageaccount*.queue.core.windows.net`
- Azure Files: `https://*mystorageaccount*.file.core.windows.net`

### NOTE

Block blob and blob storage accounts expose only the Blob service endpoint.

Construct the URL for accessing an object in a storage account by appending the object's location in the storage account to the endpoint. For example, a blob address might have this format:

`http://mystorageaccount.blob.core.windows.net/mycontainer/myblob`.

You can also configure your storage account to use a custom domain for blobs. For more information, see [Configure a custom domain name for your Azure Storage account](#).

## Control access to account data

By default, the data in your account is available only to you, the account owner. You have control over who may access your data and what permissions they have.

Every request made against your storage account must be authorized. At the level of the service, the request must include a valid *Authorization* header. Specifically, this header includes all of the information necessary for the service to validate the request before executing it.

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

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

#### NOTE

Authenticating users or applications using Azure AD credentials provides superior security and ease of use over other means of authorization. While you can continue to use Shared Key authorization with your applications, using Azure AD circumvents the need to store your account access key with your code. You can also continue to use shared access signatures (SAS) to grant fine-grained access to resources in your storage account, but Azure AD offers similar capabilities without the need to manage SAS tokens or worry about revoking a compromised SAS.

Microsoft recommends using Azure AD authorization for your Azure Storage blob and queue applications when possible.

## Copying data into a storage account

Microsoft provides utilities and libraries 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.

When you upgrade to a general-purpose v2 account from a general-purpose v1 or Blob storage account, your data is automatically migrated. Microsoft recommends this pathway for upgrading your account. However, if you decide to move data from a general-purpose v1 account to a Blob storage account, then you'll migrate your data manually, using the tools and libraries described below.

### AzCopy

AzCopy is a Windows command-line utility designed for high-performance copying of data to and from Azure Storage. You can use AzCopy to copy data into a Blob storage account from an existing general-purpose storage account, or to upload data from on-premises storage devices. For more information, see [Transfer data with the AzCopy Command-Line Utility](#).

### Data movement library

The Azure Storage data movement library for .NET is based on the core data movement framework that powers AzCopy. The library is designed for high-performance, reliable, and easy data transfer operations similar to AzCopy. You can use the data movement library to take advantage of AzCopy features natively. For more information, see [Azure Storage Data Movement Library for .NET](#)

### REST API or client library

You can create a custom application to migrate your data from a general-purpose v1 storage account into a Blob storage account. Use one of the Azure client libraries or the Azure storage services REST API. Azure Storage provides rich client libraries for multiple languages and platforms like .NET, Java, C++, Node.js, PHP, Ruby, and Python. The client libraries offer advanced capabilities such as retry logic, logging, and parallel uploads. You can also develop directly against the REST API, which can be called by any language that makes HTTP/HTTPS requests.

For more information about the Azure Storage REST API, see [Azure Storage Services REST API Reference](#).

## IMPORTANT

Blobs encrypted using client-side encryption store encryption-related metadata with the blob. If you copy a blob that is encrypted with client-side encryption, ensure that the copy operation preserves the blob metadata, and especially the encryption-related metadata. If you copy a blob without the encryption metadata, the blob content cannot be retrieved again. For more information regarding encryption-related metadata, see [Azure Storage Client-Side Encryption](#).

## Storage account billing

You're billed for Azure Storage 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.
- **Storage Capacity** refers to how much of your storage account allotment you're using to store data.
- **Replication** 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.

## Next steps

- [Create a storage account](#)
- [Create a block blob storage account](#)

# Security recommendations for Blob storage

4/22/2020 • 7 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 what Microsoft does to fulfill service provider responsibilities, read [Shared responsibilities for cloud computing](#).

Some of the recommendations included in this article can be automatically monitored by Azure Security Center. Azure Security Center is the first line of defense in protecting your resources in Azure. For information on Azure Security Center, see the [What is Azure Security Center?](#).

Azure Security Center 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 Azure Security Center recommendations, see [Security recommendations in Azure Security Center](#).

## Data protection

RECOMMENDATION	COMMENTS	SECURITY CENTER
Use the Azure Resource Manager deployment model	Create new storage accounts using the Azure Resource Manager deployment model for important security enhancements, including superior access control (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 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	SECURITY CENTER
Enable advanced threat protection for all of your storage accounts	Advanced threat protection for Azure 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 Azure Security Center 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="#">Advanced threat protection for Azure Storage</a> .	Yes
Turn on soft delete for blob data	Soft delete enables you to recover blob data after it has been deleted. For more information on soft delete, see <a href="#">Soft delete for Azure Storage blobs</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> .	-
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	SECURITY CENTER
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 Azure blobs and queues using Azure Active Directory</a> .	-
Keep in mind the principle of least privilege when assigning permissions to an Azure AD security principal via 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.	-

RECOMMENDATION	COMMENTS	SECURITY CENTER
Use a user delegation SAS to grant limited access to blob data to clients	<p>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>.</p>	-
Secure your account access keys with Azure Key Vault	<p>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>.</p>	-
Regenerate your account keys periodically	<p>Rotating the account keys periodically reduces the risk of exposing your data to malicious actors.</p>	-
Keep in mind the principle of least privilege when assigning permissions to a SAS	<p>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.</p>	-
Have a revocation plan in place for any SAS that you issue to clients	<p>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>.</p>	-
If a service SAS is not associated with a stored access policy, then set the expiry time to one hour or less	<p>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.</p>	-
Limit anonymous public read access to containers and blobs	<p>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.</p>	-

# Networking

RECOMMENDATION	COMMENTS	SECURITY CENTER
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="#">Azure File Sync proxy and firewall settings</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="#">Azure File Sync proxy and firewall settings</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

## Logging/Monitoring

RECOMMENDATION	COMMENTS	SECURITY CENTER
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="#">Azure Storage analytics logging</a> .	-

## Next steps

- [Azure security documentation](#)
- [Secure development documentation](#).

# Azure Storage encryption for data at rest

4/16/2020 • 2 minutes to read • [Edit Online](#)

Azure Storage automatically encrypts 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.

## About Azure Storage 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](#).

## About encryption key management

Data in a new storage account is encrypted with 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. If you choose to manage encryption with your own keys, you have two options:

- You can specify a *customer-managed key* with Azure Key Vault to use for encrypting and decrypting data in Blob storage and in Azure Files.<sup>1,2</sup> For more information about customer-managed keys, see [Use customer-managed keys with Azure Key Vault to manage 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 \(preview\)](#).

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

	MICROSOFT-MANAGED KEYS	CUSTOMER-MANAGED KEYS	CUSTOMER-PROVIDED KEYS
Encryption/decryption operations	Azure	Azure	Azure

	MICROSOFT-MANAGED KEYS	CUSTOMER-MANAGED KEYS	CUSTOMER-PROVIDED KEYS
Azure Storage services supported	All	Blob storage, Azure Files <sup>1,2</sup>	Blob storage
Key storage	Microsoft key store	Azure Key Vault	Customer's own key store
Key rotation responsibility	Microsoft	Customer	Customer
Key control	Microsoft	Customer	Customer

<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](#).

## Next steps

- [What is Azure Key Vault?](#)
- [Configure customer-managed keys for Azure Storage encryption from the Azure portal](#)
- [Configure customer-managed keys for Azure Storage encryption from PowerShell](#)
- [Configure customer-managed keys for Azure Storage encryption from Azure CLI](#)

# Use customer-managed keys with Azure Key Vault to manage Azure Storage encryption

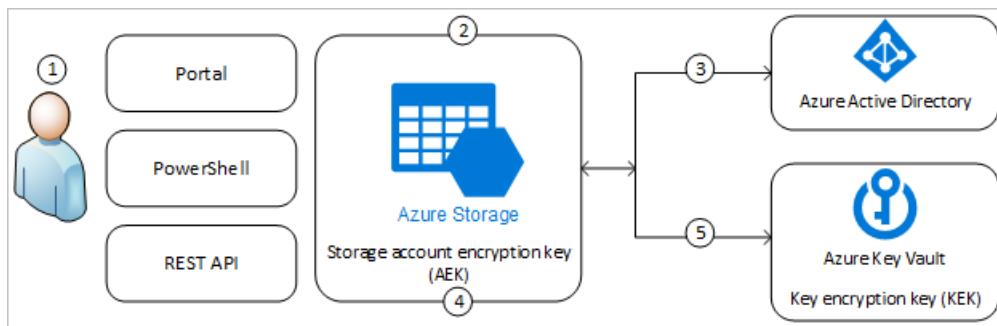
4/16/2020 • 5 minutes to read • [Edit Online](#)

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 Azure Key Vault to store your customer-managed keys. You can either create your own keys and store them in a key vault, or you can use the Azure Key Vault APIs to generate keys. The storage account and the key vault must be in the same region and in the same Azure Active Directory (Azure AD) tenant, but they can be in different subscriptions. For more information about Azure Key Vault, see [What is Azure Key Vault?](#).

## About customer-managed keys

The following diagram shows how Azure Storage uses Azure Active Directory and Azure Key Vault 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 the managed identity that's 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 that's associated with the storage account to authenticate access to Azure Key Vault via Azure Active Directory.
4. Azure Storage wraps the account encryption key with the customer 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.

## Create an account that supports customer-managed keys for queues and tables

Data stored in the Queue and Table services 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 at the time that you create the storage account to be included in this protection.

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 the Blob and File services 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

Customer-managed keys can be enabled only on existing storage accounts. The key vault must be provisioned with access policies that grant key permissions to the managed identity that is associated with the storage account. The managed identity is available only after the storage account is created.

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. Enabling customer-managed keys does not impact performance, and takes effect immediately.

When you modify the key being used for Azure Storage encryption by enabling or disabling customer-managed keys, updating the key version, or specifying a different key, then the encryption of the root key changes, but the data in your Azure Storage account does not need to be re-encrypted.

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.

To learn how to use customer-managed keys with Azure Key Vault for Azure Storage encryption, see one of these articles:

- [Configure customer-managed keys with Key Vault for Azure Storage encryption from the Azure portal](#)
- [Configure customer-managed keys with Key Vault for Azure Storage encryption from PowerShell](#)
- [Configure customer-managed keys with Key Vault for Azure Storage encryption from Azure CLI](#)

## IMPORTANT

Customer-managed keys rely on managed identities for Azure resources, a feature of Azure AD. Managed identities do not currently support cross-directory 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 directory 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](#).

## Store customer-managed keys in Azure Key Vault

To enable customer-managed keys on a storage account, you must use an Azure Key Vault to store your keys. You must enable both the **Soft Delete** and **Do Not Purge** properties on the key vault.

Only 2048-bit RSA and RSA-HSM keys are supported with Azure Storage encryption. For more information about keys, see [Key Vault keys](#) in [About Azure Key Vault keys, secrets and certificates](#).

## Rotate customer-managed keys

You can rotate a customer-managed key in Azure Key Vault according to your compliance policies. When the key is rotated, you must update the storage account to use the new key version URI. To learn how to update the storage account to use a new version of the key in the Azure portal, see the section titled [Update the key version](#) in [Configure customer-managed keys for Azure Storage by using the Azure portal](#).

Rotating the key does not trigger re-encryption of data in the storage account. There is no further action required from the user.

## 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

- [Configure customer-managed keys with Key Vault for Azure Storage encryption from the Azure portal](#)
- [Configure customer-managed keys with Key Vault for Azure Storage encryption from PowerShell](#)
- [Configure customer-managed keys with Key Vault for Azure Storage encryption from Azure CLI](#)
- [Azure Storage encryption for data at rest](#)

# Provide an encryption key on a request to Blob storage (preview)

3/15/2020 • 2 minutes to read • [Edit Online](#)

Clients making requests against Azure Blob storage have the option to provide an encryption key on a per-request basis (preview). 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 does not 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 is not 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 can have its own encryption 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. Must be AES256.

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 re-upload 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.

## Next steps

- [Specify a customer-provided key on a request to Blob storage with .NET](#)
- [Azure Storage encryption for data at rest](#)

# Authorizing access to data in Azure Storage

4/17/2020 • 2 minutes to read • [Edit Online](#)

Each time you access data in your storage account, your client makes a request over HTTP/HTTPS to Azure Storage. Every request to a secure resource must be authorized, so that the service ensures that the client has the permissions required to access the data.

The following table describes the options that Azure Storage offers for authorizing access to resources:

	SHARED KEY (STORAGE ACCOUNT KEY)	SHARED ACCESS SIGNATURE (SAS)	AZURE ACTIVE DIRECTORY (AZURE AD)	ON-PREMISES ACTIVE DIRECTORY DOMAIN SERVICES (PREVIEW)	ANONYMOUS PUBLIC READ ACCESS
Azure Blobs	Supported	Supported	Supported	Not supported	Supported
Azure Files (SMB)	Supported	Not supported	Supported, only with AAD Domain Services	Supported, credentials must be synced to Azure AD	Not supported
Azure Files (REST)	Supported	Supported	Not supported	Not supported	Not supported
Azure Queues	Supported	Supported	Supported	Not Supported	Not supported
Azure Tables	Supported	Supported	Not supported	Not supported	Not supported

Each authorization option is briefly described below:

- **Azure Active Directory (Azure AD) integration** for blobs, and queues. Azure AD provides role-based access control (RBAC) for control over a client's access to resources in a storage account. For more information regarding Azure AD integration for blobs and queues, see [Authorize access to Azure blobs and queues using Azure Active Directory](#).
- **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 RBAC for fine-grained control over a client's access to Azure Files resources in a storage account. For more information regarding Azure Files authentication using domain services, refer to the [overview](#).
- **On-premises Active Directory Domain Services (AD DS, or on-premises AD DS) authentication (preview)** 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 RBAC for share level access control and NTFS DACLs for directory/file level permission enforcement. For more information regarding Azure Files authentication using domain services, refer to the [overview](#).
- **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](#).
- **Shared access signatures** for blobs, files, queues, and tables. Shared access signatures (SAS) provide

limited delegated access to resources in a storage account. Adding constraints on the time interval for which the signature is valid or on permissions it grants provides flexibility in managing access. For more information, see [Using shared access signatures \(SAS\)](#).

- **Anonymous public read access** for containers and blobs. Authorization is not required. For more information, see [Manage anonymous read access to containers and blobs](#).

By default, all resources in Azure Storage are secured, and are available only to the account owner. Although you can use any of the authorization strategies outlined above to grant clients access to resources in your storage account, Microsoft recommends using Azure AD when possible for maximum security and ease of use.

## Next steps

- [Authorize access to Azure blobs and queues using Azure Active Directory](#)
- [Authorize with Shared Key](#)
- [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#)

# Authorize access to blobs and queues using Azure Active Directory

2/24/2020 • 8 minutes to read • [Edit Online](#)

Azure Storage supports using Azure Active Directory (Azure AD) to authorize requests to Blob and Queue storage. With Azure AD, you can use role-based access control (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 Blob or Queue storage.

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 and queue applications when possible to minimize potential security vulnerabilities inherent in Shared Key.

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](#).

Azure Files supports authorization with AD (preview) or Azure AD DS (GA) over SMB for domain-joined VMs only. To learn about using AD (preview) or Azure AD DS (GA) over SMB for Azure Files, see [Overview of Azure Files identity-based authentication support for SMB access](#).

Authorization with Azure AD is not supported for Azure Table storage. Use Shared Key to authorize requests to Table storage.

## Overview of Azure AD for blobs and queues

When a security principal (a user, group, or application) attempts to access a blob or queue 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 or Queue 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 blobs or queues. To learn how to authorize requests made by a managed identity to the Azure Blob or Queue service, see [Authorize access to blobs and queues with Azure Active Directory and managed identities for Azure Resources](#).

The authorization step requires that one or more RBAC roles be assigned to the security principal. Azure Storage provides RBAC roles that encompass common sets of permissions for blob and queue data. The roles that are assigned to a security principal determine the permissions that the principal will have. To learn more about assigning RBAC roles for Azure Storage, see [Manage access rights to storage data with RBAC](#).

Native applications and web applications that make requests to the Azure Blob or Queue service can also authorize access with Azure AD. To learn how to request an access token and use it to authorize requests for blob or queue data, see [Authorize access to Azure Storage with Azure AD from an Azure Storage application](#).

# Assign RBAC roles for access rights

Azure Active Directory (Azure AD) authorizes access rights to secured resources through [role-based access control \(RBAC\)](#). Azure Storage defines a set of built-in RBAC roles that encompass common sets of permissions used to access blob and queue data. You can also define custom roles for access to blob and queue data.

When an RBAC role is assigned to an Azure AD security principal, Azure grants access to those resources for that security principal. Access can be scoped to the level of the subscription, the resource group, the storage account, or an individual container or queue. An Azure AD security principal may be a user, a group, an application service principal, or a [managed identity for Azure resources](#).

## Built-in RBAC roles for blobs and queues

Azure provides the following built-in RBAC roles for authorizing access to blob and queue data using Azure AD and OAuth:

- [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 Queue Data Contributor](#): Use to grant read/write/delete permissions to Azure queues.
- [Storage Queue Data Reader](#): Use to grant read-only permissions to Azure queues.
- [Storage Queue Data Message Processor](#): Use to grant peek, retrieve, and delete permissions to messages in Azure Storage queues.
- [Storage Queue Data Message Sender](#): Use to grant add permissions to messages in Azure Storage queues.

For detailed information about built-in RBAC 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 RBAC roles](#), and [Azure AD roles](#).

### NOTE

RBAC role assignments may take up to five minutes to propagate.

Only roles explicitly defined for data access permit a security principal to access blob or queue data. 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 or queue data within that account.

Access to blob or queue data in the Azure portal can be authorized using either your Azure AD account or the storage account access key. For more information, see [Use the Azure portal to access blob or queue data](#).

To learn how to assign a built-in RBAC role to a security principal, see one of the following articles:

- [Grant access to Azure blob and queue data with RBAC in the Azure portal](#)
- [Grant access to Azure blob and queue data with RBAC using Azure CLI](#)
- [Grant access to Azure blob and queue data with RBAC using PowerShell](#)

For more information about how built-in roles are defined for Azure Storage, see [Understand role definitions](#). For information about creating custom RBAC roles, see [Create custom roles for Azure Role-Based Access Control](#).

## Access permissions for data operations

For details on the permissions required to call specific Blob or Queue service operations, see [Permissions for calling blob and queue data operations](#).

## Resource scope

Before you assign an 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.

The following list describes the levels at which you can scope access to Azure blob and queue resources, starting 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.
- **An individual queue.** At this scope, a role assignment applies to messages in the queue, as well as queue properties and metadata.
- **The storage account.** At this scope, a role assignment applies to all containers and their blobs, or to all queues and their messages.
- **The resource group.** At this scope, a role assignment applies to all of the containers or queues in all of the storage accounts in the resource group.
- **The subscription.** At this scope, a role assignment applies to all of the containers or queues in all of the storage accounts in all of the resource groups in the subscription.

### IMPORTANT

If your subscription includes an Azure DataBricks namespace, roles that are scoped to the subscription will not grant access to blob and queue data. Scope roles to the resource group, storage account, or container or queue instead.

## Access data with an Azure AD account

Access to blob or queue 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 and queue data in an Azure storage account. Which authorization scheme the Azure portal uses depends on the RBAC roles that are assigned to you.

When you attempt to access blob or queue data, the Azure portal first checks whether you have been assigned an RBAC 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 and queue 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 or queue data from the Azure portal using your Azure AD account, you need permissions to access blob and queue 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 and queue 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 [Grant access to Azure blob and queue data with RBAC in the Azure portal](#).

The Azure portal indicates which authorization scheme is in use when you navigate to a container or queue. For more information about data access in the portal, see [Use the Azure portal to access blob or queue data](#).

### 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 [Run Azure CLI or PowerShell commands with Azure AD credentials to access blob or queue data](#).

## Next steps

- [Authorize access to blobs and queues with Azure Active Directory and managed identities for Azure Resources](#)
- [Authorize with Azure Active Directory from an application for access to blobs and queues](#)
- [Azure Storage support for Azure Active Directory based access control generally available](#)

# Grant limited access to Azure Storage resources using shared access signatures (SAS)

2/10/2020 • 11 minutes to read • [Edit Online](#)

A shared access signature (SAS) provides secure delegated access to resources in your storage account without compromising the security of your data. With a SAS, you have granular control over how a client can access your data. You can control what resources the client may access, what permissions they have on those resources, and how long the SAS is valid, among other parameters.

## Types of shared access signatures

Azure Storage supports three types of shared access signatures:

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

For more information about the user delegation SAS, see [Create a user delegation SAS \(REST API\)](#).

- **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. Additionally, with the account SAS, you can delegate access to operations that apply at the level of the service, such as **Get/Set Service Properties** and **Get Service Stats** operations. You can also delegate access to read, write, and delete operations on blob containers, tables, queues, and file shares that are not permitted with a service SAS.

For more information about the account SAS, [Create an account SAS \(REST API\)](#).

### 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.

A shared access signature can take one of two forms:

- **Ad hoc SAS:** When you create an ad hoc SAS, the start time, expiry time, and permissions for the SAS are all specified in the SAS URI (or implied, if start time is omitted). 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 and 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.

### SAS signature

You can sign a SAS in one of two ways:

- With 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 user delegation key and create the SAS, an Azure AD security principal must be assigned a role-based access control (RBAC) role that includes the

**Microsoft.Storage/storageAccounts/blobServices/generateUserDelegationKey** action. For detailed information about RBAC roles with permissions to get the user delegation key, see [Create a user delegation SAS \(REST API\)](#).

- With the storage 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.

### 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.

When a client application provides a SAS URI to Azure Storage as part of a request, the service checks the SAS parameters and signature to verify that it is valid for authorizing the request. 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:



## When to use a shared access signature

Use a SAS when you want to provide secure access to resources in your storage account to any client who does not otherwise have permissions to those resources.

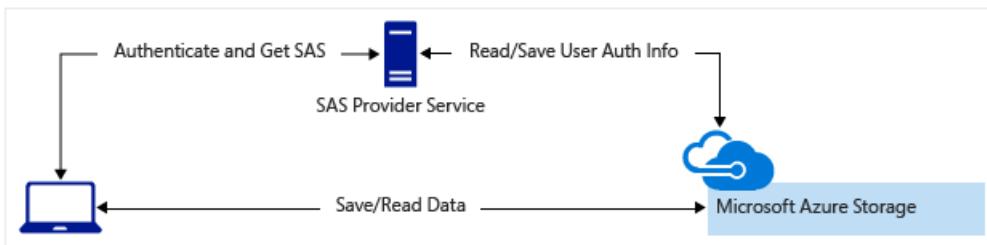
A common scenario where a SAS is useful is a service where users read and write their own data to your

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 has the advantage of allowing 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, they can access storage account resources directly with the permissions 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, while 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 must use a SAS to authorize access to the source blob. 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 must use a SAS to authorize access to the source file. 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 to authorize access to the source object, 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 and then use it just as the intended user could have, potentially compromising sensitive data or allowing for data corruption by the malicious user.

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

to every client for access.

- **Use Azure Monitor and Azure Storage logs to monitor your application.** You can use Azure Monitor and storage analytics logging to observe any spike in authorization failures due to an outage in your SAS provider service or to the inadvertent removal of a stored access policy. For more information, see [Azure Storage metrics in Azure Monitor](#) and [Azure Storage Analytics logging](#).

## 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 \(REST API\)](#)

# Manage anonymous read access to containers and blobs

12/5/2019 • 4 minutes to read • [Edit Online](#)

You can enable anonymous, public read access to a container and its blobs in Azure Blob storage. By doing so, you can grant read-only access to these resources without sharing your account key, and without requiring a shared access signature (SAS).

Public read access is best for scenarios where you want certain blobs to always be available for anonymous read access. For more fine-grained control, you can create a shared access signature. Shared access signatures enable you to provide restricted access using different permissions, for a specific time period. For more information about creating shared access signatures, see [Using shared access signatures \(SAS\) in Azure Storage](#).

## Grant anonymous users permissions to containers and blobs

By default, a container and any blobs within it may be accessed only by a user that has been given appropriate permissions. To grant anonymous users read access to a container and its blobs, you can set the container public access level. When you grant public access to a container, then anonymous users can read blobs within a publicly accessible container without authorizing the request.

You can configure a container with the following permissions:

- **No public read access:** The container and its blobs can be accessed only by the storage account owner. This 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. Anonymous clients cannot enumerate the blobs within the container.
- **Public read access for container and its blobs:** All container and blob data can be read by anonymous request. Clients can enumerate blobs within the container by anonymous request, but cannot enumerate containers within the storage account.

### Set container public access level in the Azure portal

From the [Azure portal](#), you can update the public access level for one or more containers:

1. Navigate to your storage account overview in the Azure portal.
2. Under **Blob service** on the menu blade, select **Blobs**.
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.
5. 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 following screenshot shows how to change the public access level for the selected containers.

The screenshot shows the Azure Storage Explorer interface for a storage account named 'storagesamples - Blobs'. On the left, there's a sidebar with various navigation options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Data transfer, Events, Storage Explorer (preview), Settings (Access keys, Geo-replication, CORS, Configuration), and a Search bar. The main area displays a list of containers: sample-container1, sample-container2, sample-container3, sample-container4 (which is selected, indicated by a checked checkbox), and sample-container5. At the top right, there are buttons for Container, Refresh, Delete, and Change access level, with the 'Change access level' button being highlighted with a red box.

#### NOTE

You cannot change the public access level for an individual blob. Public access level is set only at the container level.

### Set container public access level with .NET

To set permissions for a container using the Azure Storage client library for .NET, first retrieve the container's existing permissions by calling one of the following methods:

- [GetPermissions](#)
- [GetPermissionsAsync](#)

Next, set the **PublicAccess** property on the [BlobContainerPermissions](#) object that is returned by the [GetPermissions](#) method.

Finally, call one of the following methods to update the container's permissions:

- [SetPermissions](#)
- [SetPermissionsAsync](#)

The following example sets the container's permissions to full public read access. To set permissions to public read access for blobs only, set the **PublicAccess** property to [BlobContainerPublicAccessType.Blob](#). To remove all permissions for anonymous users, set the property to [BlobContainerPublicAccessType.Off](#).

```
private static async Task SetPublicContainerPermissions(CloudBlobContainer container)
{
 BlobContainerPermissions permissions = await container.GetPermissionsAsync();
 permissions.PublicAccess = BlobContainerPublicAccessType.Container;
 await container.SetPermissionsAsync(permissions);

 Console.WriteLine("Container {0} - permissions set to {1}", container.Name, permissions.PublicAccess);
}
```

# Access containers and blobs anonymously

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.

## 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.

```
public static void CreateAnonymousBlobClient()
{
 // Create the client object using the Blob storage endpoint for your account.
 CloudBlobClient blobClient = new CloudBlobClient(
 new Uri(@"https://storagesamples.blob.core.windows.net"));

 // Get a reference to a container that's available for anonymous access.
 CloudBlobContainer container = blobClient.GetContainerReference("sample-container");

 // Read the container's properties.
 // Note this is only possible when the container supports full public read access.
 container.FetchAttributes();
 Console.WriteLine(container.Properties.LastModified);
 Console.WriteLine(container.Properties.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.

```
public static void ListBlobsAnonymously()
{
 // Get a reference to a container that's available for anonymous access.
 CloudBlobContainer container = new CloudBlobContainer(
 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 (IListBlobItem blobItem in container.ListBlobs())
 {
 Console.WriteLine(blobItem.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:

```
public static void DownloadBlobAnonymously()
{
 CloudBlockBlob blob = new CloudBlockBlob(
 new Uri(@"https://storagesamples.blob.core.windows.net/sample-container/logfile.txt"));
 blob.DownloadToFile(@"C:\Temp\logfile.txt", FileMode.Create);
}
```

## Next steps

- [Authorizing access to Azure Storage](#)
- [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#)

- Blob Service REST API

# Use the Azure Storage resource provider to access management resources

1/14/2020 • 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. RBAC provides fine-grained control over permissions to both types of resources. When you assign an RBAC 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 built-in RBAC role, see [Built-in roles for Azure resources](#).

Azure Storage supports using Azure AD to authorize requests against Blob and Queue storage. For information about RBAC roles for blob and queue data operations, see [Authorize access to blobs and queues using Active Directory](#).

## Assign management permissions with role-based access control (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 role-based access control (RBAC).

When you assign an RBAC 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 RBAC role to a security principal by using the [Azure portal](#), the [Azure CLI tools](#), [PowerShell](#), or the [Azure Storage resource provider REST API](#).

For more information about RBAC, see [What is role-based access control \(RBAC\) for Azure resources?](#) and [Classic subscription administrator roles, Azure RBAC 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:

RBAC 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 assign 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 custom RBAC roles for access to management resources. For more information about custom roles, see [Custom roles for Azure resources](#).

## 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 role-based access control \(RBAC\) for Azure resources?](#)
- [Scalability targets for the Azure Storage resource provider](#)

# Store business-critical blob data with immutable storage

3/13/2020 • 15 minutes to read • [Edit Online](#)

Immutable storage for Azure Blob storage enables users to store business-critical data objects in a WORM (Write Once, Read Many) state. This state makes the data non-erasable and non-modifiable for a user-specified interval. For the duration of the retention interval, blobs can be created and read, but cannot be modified or deleted. Immutable storage is available for general-purpose v1, general-purpose v2, BlobStorage, and BlockBlobStorage accounts in all Azure regions.

For information about how to set and clear legal holds or create a time-based retention policy using the Azure portal, PowerShell, or Azure CLI, see [Set and manage immutability policies for Blob storage](#).

## NOTE

This feature 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](#).

## About immutable Blob storage

Immutable storage helps healthcare organization, financial institutions, and related industries—particularly broker-dealer organizations—to store data securely. Immutable storage can also 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. A technical whitepaper by Cohasset Associates details how Immutable storage addresses these regulatory requirements is downloadable via the [Microsoft Service Trust Portal](#). The [Azure Trust Center](#) contains detailed information about our compliance certifications.
- **Secure document retention:** Immutable storage for Azure Blob storage ensures that data can't be modified or deleted by any user, including users with account administrative privileges.
- **Legal hold:** Immutable storage for Azure Blob storage 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.

Immutable storage supports the following features:

- **Time-based retention policy support:** Users can set policies to store data for a specified interval. When a time-based retention policy is set, blobs can be created and read, but not modified or deleted. After the retention period has expired, blobs can be deleted but not overwritten.
- **Legal hold policy support:** If the retention interval is not known, users can set legal holds to store immutable data until the legal hold is cleared. When a legal hold policy is set, blobs can be created and read, but not modified or deleted. Each legal hold is associated with a user-defined alphanumeric tag (such as a case ID, event name, etc.) that is used as an identifier string.

- **Support for all blob tiers:** WORM policies are independent of the Azure Blob storage tier and apply to all the tiers: hot, cool, and archive. Users can transition data to the most cost-optimized tier for their workloads while maintaining data immutability.
- **Container-level configuration:** Users can configure time-based retention policies and legal hold tags at the container level. By using simple container-level settings, users can create and lock time-based retention policies, extend retention intervals, set and clear legal holds, and more. These policies apply to all the blobs in the container, both existing and new.
- **Audit logging support:** Each container includes a policy audit log. It shows up to seven time-based retention commands for locked time-based retention policies and contains the user ID, command type, time stamps, and retention interval. For legal holds, the log contains the user ID, command type, time stamps, and legal hold tags. This log is retained for the lifetime of the policy, in accordance with the SEC 17a-4(f) regulatory guidelines. The [Azure Activity Log](#) shows a more comprehensive log of all the control plane activities; while enabling [Azure Diagnostic Logs](#) retains and shows data plane operations. It is the user's responsibility to store those logs persistently, as might be required for regulatory or other purposes.

## How it works

Immutable storage for Azure Blob storage supports two types of WORM or immutable policies: time-based retention and legal holds. When a time-based retention policy or legal hold is applied on 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, the immutable policy is confirmed and any overwrite or delete operations in the immutable container are not allowed.

Container and storage account deletion are also not permitted if there are any blobs in a container that are protected by a legal hold or a locked time-based policy. A legal hold policy will protect against blob, container, and storage account deletion. Both unlocked and locked time-based policies will protect against blob deletion for the specified time. Both unlocked and locked time-based policies will protect against container deletion only if at least one blob exists within the container. Only a container with *locked* time-based policy will protect against storage account deletions; containers with unlocked time-based policies do not offer storage account deletion protection nor compliance.

For more information on how to set and lock time-based retention policies, see [Set and manage immutability policies for Blob storage](#).

## Time-based retention policies

### 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. We recommend that you lock the policy in a reasonable amount of time, typically less than 24 hours. The initial state of an applied time-based retention policy is *unlocked*, allowing you to test the feature and make changes to the policy before you lock it. While the *unlocked* state provides immutability protection, we don't recommend using the *unlocked* state for any purpose other than short-term feature trials.

When a time-based retention policy is applied on a container, all blobs in the container will stay in the immutable state for the duration of the *effective* retention period. The effective retention period for blobs is equal to the difference between the blob's **creation time** and the user-specified retention interval. Because users can extend the retention interval, 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 the *testblob2* is five years from the time of its creation.

An unlocked time-based retention policy is recommended only for feature testing and a policy must be locked in order to be compliant with SEC 17a-4(f) and other regulatory compliance. Once a time-based retention policy is locked, the policy cannot be removed and a maximum of five increases to the effective retention period is allowed.

The following limits apply to retention policies:

- For a storage account, the maximum number of containers with locked time-based immutable policies is 10,000.
- The minimum retention interval is 1 day. The maximum is 146,000 days (400 years).
- For a container, the maximum number of edits to extend a retention interval for locked time-based immutable policies is 5.
- For a container, a maximum of seven time-based retention policy audit logs are retained for a locked policy.

### Allow protected append blobs writes

Append blobs are comprised of data blocks 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 `allowProtectedAppendWrites` setting that allows for writing new blocks to an append blob while maintaining immutability protection and compliance. If enabled, you are allowed to create an append blob directly in the policy protected container and continue to add new blocks of data to the end of existing append blobs using the *AppendBlock* API. Only new blocks can be added and 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 still stay 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 `allowProtectedAppendWrites` 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, the effective retention period for the *logblob1* is 100 days from today (the time of its last append + 90 days).

Unlocked time-based retention policies allow the `allowProtectedAppendWrites` setting to be enabled and disabled at any time. Once the time-based retention policy is locked, the `allowProtectedAppendWrites` setting cannot be changed.

Legal hold policies cannot enable `allowProtectedAppendWrites` and any legal holds will nullify the 'allowProtectedAppendWrites' property. If a legal hold is applied to a time-based retention policy with `allowProtectedAppendWrites` enabled, the *AppendBlock* API will fail until the legal hold is lifted.

## Legal holds

Legal holds are temporary holds that can be used for legal investigation purposes or general protection policies. Each legal hold policy needs to be associated with one or more tags. Tags are used as a named identifier, such as a case ID or event, to categorize and describe the purpose of the hold.

A container can have both a legal hold and a time-based retention policy at the same time. All blobs in that container stay in the immutable state until all legal holds are cleared, even if their effective retention period has expired. Conversely, a blob stays in an immutable state until the effective retention period expires, even though all legal holds have been cleared.

The following limits apply to 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 10.
- 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 10 legal hold policy audit logs are retained for the duration of the policy.

## Scenarios

The following table shows the types of Blob storage operations that are disabled for the different immutable scenarios. For more information, see the [Azure Blob Service REST API](#) documentation.

SCENARIO	BLOB STATE	BLOB OPERATIONS DENIED	CONTAINER AND ACCOUNT PROTECTION
Effective retention interval on the blob has not yet expired and/or legal hold is set	Immutable: both delete and write-protected	Put Blob <sup>1</sup> , Put Block <sup>1</sup> , Put Block List <sup>1</sup> , Delete Container, Delete Blob, Set Blob Metadata, Put Page, Set Blob Properties, Snapshot Blob, Incremental Copy Blob, Append Block <sup>2</sup>	Container deletion denied; Storage Account deletion denied
Effective retention interval on the blob has expired and no legal hold is set	Write-protected only (delete operations are allowed)	Put Blob <sup>1</sup> , Put Block <sup>1</sup> , Put Block List <sup>1</sup> , Set Blob Metadata, Put Page, Set Blob Properties, Snapshot Blob, Incremental Copy Blob, Append Block <sup>2</sup>	Container deletion denied if at least 1 blob exists within protected container; Storage Account deletion denied only for <i>locked</i> time-based policies
No WORM policy applied (no time-based retention and no legal hold tag)	Mutable	None	None

<sup>1</sup> The blob service allows these operations to create a new blob once. All subsequent overwrite operations on an existing blob path in an immutable container are not allowed.

<sup>2</sup> Append Block is only allowed for time-based retention policies with the `allowProtectedAppendWrites` property enabled. For more information, see the [Allow Protected Append Blobs Writes](#) section.

## Pricing

There is no additional charge for using this feature. Immutable data is priced in the same way as mutable data. For pricing details on Azure Blob storage, see the [Azure Storage pricing page](#).

## FAQ

### Can you provide documentation of WORM compliance?

Yes. To document compliance, Microsoft retained a leading independent assessment firm that specializes in records management and information governance, Cohasset Associates, to evaluate immutable Blob storage and its

compliance with requirements specific to the financial services industry. Cohasset validated that immutable Blob storage, when used to retain time-based 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. Microsoft targeted this set of rules, as they represent the most prescriptive guidance globally for records retention for financial institutions. The Cohasset report is available in the [Microsoft Service Trust Center](#). To request a letter of attestation from Microsoft regarding WORM immutability compliance, please contact Azure support.

### **Does the feature apply to only block blobs and append blobs, or to page blobs as well?**

Immutable storage can be used with any blob type as it is set at the container level, but we recommend that you use WORM for containers that mainly store block blobs and append blobs. Existing blobs in a container will be protected by a newly set WORM policy. But any new page blobs need to be created outside the WORM container, and then copied in. Once copied into a WORM container, no further changes to a page blob are allowed. Setting a WORM policy on a container that stores VHDs (page blobs) for any active virtual machines is discouraged as it will lock the VM disk. We recommend that you thoroughly review the documentation and test your scenarios before locking any time-based policies.

### **Do I need to create a new storage account to use this feature?**

No, you can use immutable storage with any existing or newly created general-purpose v1, general-purpose v2, BlobStorage, or BlockBlobStorage accounts. General-purpose v1 storage accounts are supported but we recommend upgrading to general-purpose v2 such that you can take advantage of more features. For information on upgrading an existing general-purpose v1 storage account, see [Upgrade a storage account](#).

### **Can I apply both a legal hold and time-based retention policy?**

Yes, a container can have both a legal hold and a time-based retention policy at the same time; however, the 'allowProtectedAppendWrites' setting will not apply until the legal hold is cleared. All blobs in that container stay in the immutable state until all legal holds are cleared, even if their effective retention period has expired. Conversely, a blob stays in an immutable state until the effective retention period expires, even though all legal holds have been cleared.

### **Are legal hold policies only for legal proceedings or are there other use scenarios?**

No, Legal Hold is just the general term used for a non-time-based retention policy. It does not need to only be used for litigation-related proceedings. Legal Hold policies are useful for disabling overwrite and deletes for protecting important enterprise WORM data, where the retention period is unknown. You may use it as an enterprise policy to protect your mission critical WORM workloads or use it as a staging policy before a custom event trigger requires the use of a time-based retention policy.

### **Can I remove a *locked* time-based retention policy or legal hold?**

Only unlocked time-based retention policies can be removed from a container. Once a time-based retention policy is locked, it cannot be removed; only effective retention period extensions are allowed. Legal hold tags can be deleted. When all legal tags are deleted, the legal hold is removed.

### **What happens if I try to delete a container with a time-based retention policy or legal hold?**

The Delete Container operation will fail if at least one blob exists within the container with either a locked or unlocked time-based retention policy or if the container has a legal hold. The Delete Container operation will succeed only if no blobs exist within the container and there are no legal holds.

### **What happens if I try to delete a storage account with a container that has a time-based retention policy or legal hold?**

The storage account deletion will fail if there is at least one container with a legal hold set or a **locked** time-based policy. A container with an unlocked time-based policy does not protect against storage account deletion. You must remove all legal holds and delete all **locked** containers before you can delete the storage account. For information

on container deletion, see the preceding question. You can also apply further delete protections for your storage account with [Azure Resource Manager locks](#).

### Can I move the data across different blob tiers (hot, cool, archive) when the blob is in the immutable state?

Yes, you can use the Set Blob Tier command to move data across the blob tiers while keeping the data in the compliant immutable state. Immutable storage is supported on hot, cool, and archive blob tiers.

### What happens if I fail to pay and my retention interval has not expired?

In the case of non-payment, 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](#).

### Do you offer a trial or grace period for just trying out the feature?

Yes. When a time-based retention policy is first created, it is in an *unlocked* state. In this state, you can make any desired change to the retention interval, such as increase or decrease and even delete the policy. After the policy is locked, it stays locked until the retention interval expires. This locked policy prevents deletion and modification to the retention interval. We strongly recommend that you use the *unlocked* state only for trial purposes and lock the policy within a 24-hour period. These practices help you comply with SEC 17a-4(f) and other regulations.

### Can I use soft delete alongside Immutable blob policies?

Yes, if your compliance requirements allow for soft delete to be enabled. [Soft delete for Azure Blob storage](#) applies for all containers within a storage account regardless of a legal hold or time-based retention policy. We recommend enabling soft delete for additional protection before any immutable WORM policies are applied and confirmed.

## Next steps

- [Set and manage immutability policies for Blob storage](#)
- [Set rules to automatically tier and delete blob data with lifecycle management](#)
- [Soft delete for Azure Storage blobs](#)
- [Protect subscriptions, resource groups, and resources with Azure Resource Manager locks](#).

# Configure advanced threat protection for Azure Storage

4/16/2020 • 3 minutes to read • [Edit Online](#)

Advanced threat protection for Azure 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 [Azure Security Center](#), 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 diagnostic logs of read, write, and delete requests to Blob Storage for threat detection. To investigate the alerts from advanced threat protection, 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

Advanced threat protection for Azure Storage is currently available only for [Blob Storage](#). Account types that support advanced threat protection include general-purpose v2, block blob, and Blob storage accounts. Advanced threat protection is available in all public clouds and US government clouds, but not in other sovereign or Azure government cloud regions.

For pricing details, including a free 30 day trial, see the [Azure Security Center pricing page](#).

## Set up advanced threat protection

You can configure advanced threat protection in any of several ways, described in the following sections.

- [Portal](#)
- [Azure Security Center](#)
- [Template](#)
- [Azure Policy](#)
- [REST API](#)
- [PowerShell](#)

1. Launch the [Azure portal](#).
2. Navigate to your Azure Storage account. Under **Settings**, select **Advanced security**.
3. Select the **Settings** link on the advanced security configuration page.
4. Set **Advanced security** to **ON**.
5. Click **Save** to save the new or updated policy.

The screenshot shows the 'Advanced security' settings page for a storage account named 'storagesecuritysamples'. The left sidebar lists various security options: Access keys, Geo-replication, CORS, Configuration, Encryption, Shared access signature, Firewalls and virtual networks, Private endpoint connection..., and Advanced security (which is selected and highlighted with a red box). Below these are Blob service options: Blobs, Custom domain, and Soft delete. At the top, there are 'Settings' and 'Refresh' buttons, and a link to 'Visit Security Center'. The main area displays three sections: 'Recommendations' (0), 'Security alerts' (0), and 'Settings'. Under 'Settings', it shows 'Advanced Threat Protection: Enabled (Settings)' (also highlighted with a red box), 'Security tier: Security Center Standard', and a note about analyzing ~0 transactions/day. A section titled 'Recommendations (preview)' states that Azure Security Center monitors for vulnerabilities and recommends actions. It shows a list of three recommendations, all of which are currently closed (indicated by a checkmark and a minus sign). A button at the bottom right says 'View all recommendations in Security Center'.

## 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	00000000-0000-0000-0000-000000000000
Storage account	mystorageaccount
Storage type	Blob
Container	mycontainer
Application	myTestApplication
IP address	192.168.1.1
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 Azure Security Center's [Security alerts tile](#). Clicking on a specific alert provides details and actions for investigating the current threat and addressing future threats.

**Anonymous access**  
mystorageaccount

[Learn more](#)

### General Information

DESCRIPTION	Someone has anonymously accessed your Azure Storage account "mystorageaccount"
DETECTION TIME	Friday, August 17, 2018, 10:50:00 AM
SEVERITY	<span style="color: orange;">⚠</span> Medium
STATE	Active
ATTACHED RESOURCE	mystorageresourcee
SUBSCRIPTION	[REDACTED]
DETECTED BY	 Microsoft
ENVIRONMENT	Azure
RESOURCE TYPE	 Storage
STORAGE ACCOUNT	mystorageaccount
STORAGE TYPE	Blob
CONTAINER	mycontainer
LOCATION	Washington, United States
USER AGENT	testUserAgentHeader
IP ADDRESS	[REDACTED]
DATA CENTER	East US

## 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 the [Storage](#) section in [Threat protection for data services in Azure Security Center](#).

## Next steps

- Learn more about [Logs in Azure Storage accounts](#)
- Learn more about [Azure Security Center](#)

# Security controls for Azure Storage

4/22/2020 • 2 minutes to read • [Edit Online](#)

This article documents the security controls built into Azure Storage.

A security control is a quality or feature of an Azure service that contributes to the service's ability to prevent, detect, and respond to security vulnerabilities.

For each control, we use "Yes" or "No" to indicate whether it is currently in place for the service, "N/A" for a control that is not applicable to the service. We might also provide a note or links to more information about an attribute.

## Data protection

SECURITY CONTROL	YES/NO	NOTES
Server-side encryption at rest: Microsoft-managed keys	Yes	
Server-side encryption at rest: customer-managed keys (BYOK)	Yes	See <a href="#">Storage Service Encryption using customer-managed keys in Azure Key Vault</a> .
Column level encryption (Azure Data Services)	N/A	
Encryption in transit (such as ExpressRoute encryption, in VNet encryption, and VNet-VNet encryption)	Yes	Support standard HTTPS/TLS mechanisms. Users can also encrypt data before it is transmitted to the service.
API calls encrypted	Yes	

## Network

SECURITY CONTROL	YES/NO	NOTES
Service endpoint support	Yes	
Service tags support	Yes	See <a href="#">Azure service tags overview</a> for more information about service tags supported by Azure Storage.
VNet injection support	N/A	
Network isolation and firewall support	Yes	
Forced tunneling support	N/A	

## Monitoring & logging

SECURITY CONTROL	YES/NO	NOTES
Azure monitoring support (Log analytics, App insights, etc.)	Yes	Azure Monitor Metrics
Control and management plane logging and audit	Yes	Azure Resource Manager Activity Log
Data plane logging and audit	Yes	Service Diagnostic Logs.

## Identity

SECURITY CONTROL	YES/NO	NOTES
Authentication	Yes	Azure Active Directory, Shared key, Shared access token.
Authorization	Yes	Support Authorization via RBAC, POSIX ACLs, and SAS Tokens

## Configuration management

SECURITY CONTROL	YES/NO	NOTES
Configuration management support (versioning of configuration, etc.)	Yes	Support Resource Provider versioning through Azure Resource Manager APIs

## Next steps

- Learn more about the [built-in security controls across Azure services](#).

# Use private endpoints for Azure Storage

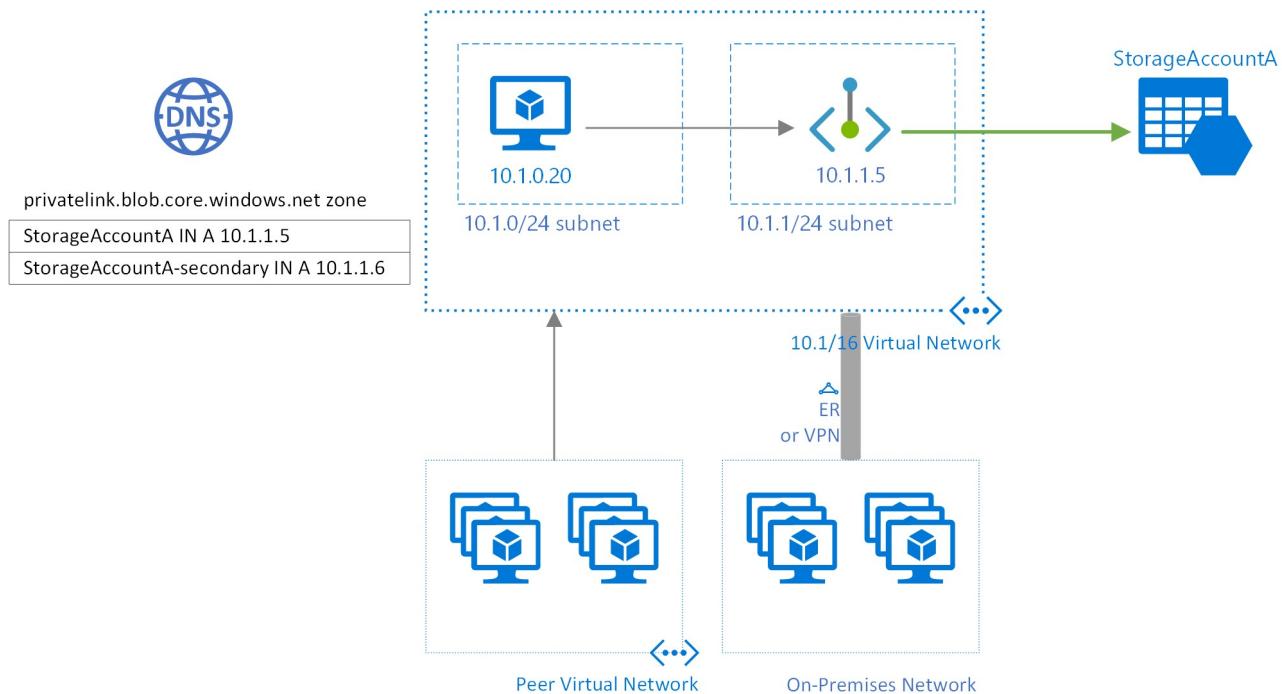
3/13/2020 • 7 minutes to read • [Edit Online](#)

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 an IP address from the VNet address space for your 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.

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.

### Private endpoints for Azure Storage

When creating the 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 service in a storage account that you need to access, namely [Blobs](#), [Data Lake Storage Gen2](#), [Files](#), [Queues](#), [Tables](#), or [Static Websites](#).

#### TIP

Create a separate private endpoint for the secondary instance of the storage service for better read performance on RA-GRS accounts.

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](#).

For more detailed information on creating a private endpoint for your storage account, refer to the following articles:

- [Connect privately to a storage account from the Storage Account experience in the Azure portal](#)
- [Create a private endpoint using the Private Link Center in the Azure portal](#)
- [Create a private endpoint using Azure CLI](#)
- [Create a private endpoint using Azure PowerShell](#)

### Connecting to private endpoints

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.

We create a [private DNS zone](#) attached to the VNet with the necessary updates for the private endpoints, by default. 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
StorageAccountA.blob.core.windows.net	CNAME	StorageAccountA.privatelink.blob.core.windows.net
StorageAccountA.privatelink.blob.core.windows.net	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
StorageAccountA.blob.core.windows.net	CNAME	StorageAccountA.privatelink.blob.core.windows.net
StorageAccountA.privatelink.blob.core.windows.net	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 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 are:

STORAGE SERVICE	ZONE NAME
Blob service	privatelink.blob.core.windows.net
Data Lake Storage Gen2	privatelink.dfs.core.windows.net
File service	privatelink.file.core.windows.net
Queue service	privatelink.queue.core.windows.net
Table service	privatelink.table.core.windows.net
Static Websites	privatelink.web.core.windows.net

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.

### **Copy Blob support**

If the storage account is protected by a firewall and the account is accessed through private endpoints, then that account cannot serve as the source of a [Copy Blob](#) operation.

### **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 instance, 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 applied to the private endpoint. 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.

## Next steps

- [Configure Azure Storage firewalls and virtual networks](#)
- [Security recommendations for Blob storage](#)

# Azure Storage redundancy

4/16/2020 • 12 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.

When deciding which redundancy option is best for your scenario, consider the tradeoffs between lower costs and higher availability and durability. 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 location that is geographically distant to the primary region, to protect against regional disasters
- Whether your application requires read access to the replicated data in the secondary region if the primary region becomes unavailable for any reason

## Redundancy in the primary region

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

- **Locally redundant storage (LRS)** copies your data synchronously three times within a single physical location in the primary region. LRS is the least expensive replication option, but is not recommended for applications requiring high availability.
- **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.

### Locally-redundant storage

Locally redundant storage (LRS) replicates your data three times within a single physical location 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 \(preview\) \(GZRS\)](#).

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.

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](#).

### Zone-redundant storage

Zone-redundant storage (ZRS) replicates your Azure Storage data 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 Azure Storage data objects of at least 99.999999999% (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 re-pointing. 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 consistency, durability, and high availability. 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 following table shows which types of storage accounts support ZRS in which regions:

Storage Account Type	Supported Regions	Supported Services
General-purpose v2 <sup>1</sup>	Asia Southeast Australia East Europe North Europe West France Central Japan East South Africa North UK South US Central US East US East 2 US West 2	Block blobs Page blobs <sup>2</sup> File shares (standard) Tables Queues
BlockBlobStorage <sup>1</sup>	Europe West US East	Block blobs only
FileStorage	Europe West US East	Azure Files only

<sup>1</sup> The archive tier is not currently supported for ZRS accounts.

<sup>2</sup> Storage accounts that contain Azure managed disks for virtual machines always use LRS. Azure unmanaged disks should also use LRS. It is possible to create a storage account for Azure unmanaged disks that uses GRS, but it is not recommended due to potential issues with consistency over asynchronous geo-replication. Neither managed nor unmanaged disks support ZRS or GZRS. For more information on managed disks, see [Pricing for Azure managed disks](#).

For information about which regions support ZRS, see [Services support by region](#) in [What are Azure Availability Zones?](#).

## Redundancy in a secondary region

For applications requiring high availability, 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.
- **Geo-zone-redundant storage (GZRS) (preview)** 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.

The primary difference between GRS and GZRS is how data is replicated in the primary region. Within the secondary location, data is always replicated synchronously three times using LRS.

With GRS or GZRS, the data in the secondary location isn't available for read or write access unless there is a failover to the secondary region. For read access to the secondary location, 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 (preview). 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 account failover \(preview\)](#).

#### 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. Azure Storage 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 Azure Storage data objects 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.

### Geo-zone-redundant storage (preview)

Geo-zone-redundant storage (GZRS) (preview) 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.

Only general-purpose v2 storage accounts support GZRS and RA-GZRS. For more information about storage account types, see [Azure storage account overview](#). GZRS and RA-GZRS support block blobs, page blobs (except for VHD disks), files, tables, and queues.

GZRS and RA-GZRS are currently available for preview in the following regions:

- Asia Southeast
- Europe North
- Europe West
- Japan East
- UK South
- US East
- US East 2
- US Central
- US West 2

Microsoft continues to enable GZRS and RA-GZRS in additional Azure regions. Check the [Azure Service Updates](#) page regularly for information about supported regions.

For information on preview pricing, refer to GZRS preview pricing for [Blobs](#), [Files](#), [Queues](#), and [Tables](#).

**IMPORTANT**

Microsoft recommends against using preview features for production workloads.

## 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. However, that data is available to be read only if the customer or Microsoft initiates a failover from the primary to secondary region. When you enable read access to the secondary region, your data is available to be read if the primary region becomes unavailable. For read access to the secondary region, enable 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 always available for read access, so you can test your application to make sure that it will read from the secondary in the event of an outage. For more information about how to design your applications for high availability, see [Designing highly available applications using read-access geo-redundant storage](#).

When read access to the secondary is enabled, your data can be read from the secondary endpoint as well as from the primary endpoint for your storage account. 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 will not 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 are 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 following table shows how durable and available your data is in a given scenario, depending on which type of redundancy is in effect for your storage account:

SCENARIO	LRS	ZRS	GRS/RA-GRS	GZRS/RA-GZRS (PREVIEW)
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	Yes
A region-wide outage occurs	No	No	Yes	Yes
Read access to data in the secondary region if the primary region becomes unavailable	No	No	Yes (with RA-GRS)	Yes (with RA-GZRS)
Percent durability of objects over a given year <sup>1</sup>	at least 99.99999999% (11 9's)	at least 99.9999999999% (12 9's)	at least 99.999999999999% (16 9's)	at least 99.99999999999999% (16 9's)
Supported storage account types <sup>2</sup>	GPv2, GPv1, BlockBlobStorage, BlobStorage, FileStorage	GPv2, BlockBlobStorage, FileStorage	GPv2, GPv1, BlobStorage	GPv2
Availability SLA for read requests <sup>1</sup>	At least 99.9% (99% for cool access tier)	At least 99.9% (99% for cool access tier)	At least 99.9% (99% for cool access tier) for GRS At least 99.99% (99.9% for cool access tier) for RA-GRS	At least 99.9% (99% for cool access tier) for GZRS At least 99.99% (99.9% for cool access tier) for RA-GZRS
Availability SLA for write requests <sup>1</sup>	At least 99.9% (99% for cool access tier)	At least 99.9% (99% for cool access tier)	At least 99.9% (99% for cool access tier)	At least 99.9% (99% for cool access tier)

<sup>1</sup> For information about Azure Storage guarantees for durability and availability, see the [Azure Storage SLA](#).

<sup>2</sup> For information for storage account types, see [Storage account overview](#).

All data for all types of storage accounts and [all tiers \(including archive\)](#) are copied according to the redundancy option for the storage account. Objects including block blobs, append blobs, page blobs, queues, tables, and files are copied.

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.

## Data integrity

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

## See also

- [Check the Last Sync Time property for a storage account](#)
- [Change the redundancy option for a storage account](#)
- [Designing highly available applications using RA-GRS Storage](#)
- [Disaster recovery and account failover \(preview\)](#)

# Disaster recovery and account failover (preview)

3/13/2020 • 12 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 (preview) 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.

## NOTE

This feature 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](#).

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 \(preview\)](#).

## NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

## 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, choose 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) (preview)** 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) (preview)** 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 RA-GRS and designed for high availability can continue to read from the secondary endpoint. Microsoft recommends RA-GRS for maximum resiliency 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.
- [Availability checklist](#): A checklist for verifying that your application implements the best design practices for high availability.
- [Designing highly available applications using RA-GRS](#): Design guidance for building applications to take advantage of RA-GRS.
- [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 [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.

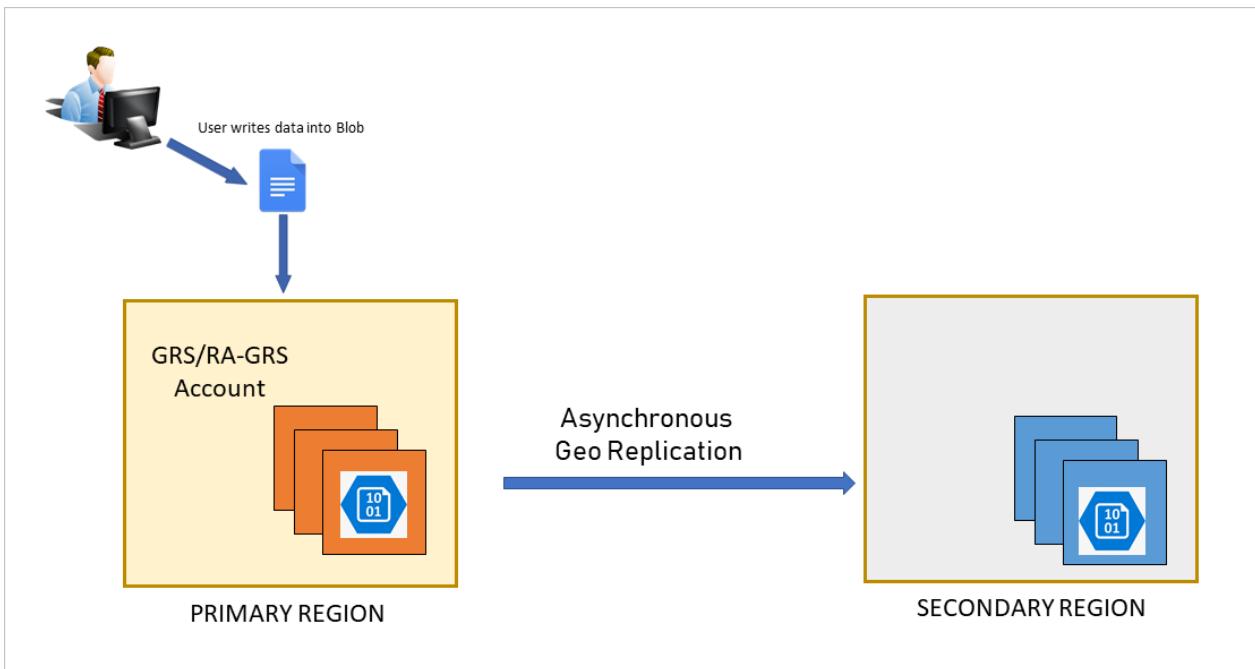
Microsoft also recommends that you design your application to prepare for the possibility of write failures. Your application should expose write failures in a way that alerts you to the possibility of an outage in the primary region.

## Understand the account failover process

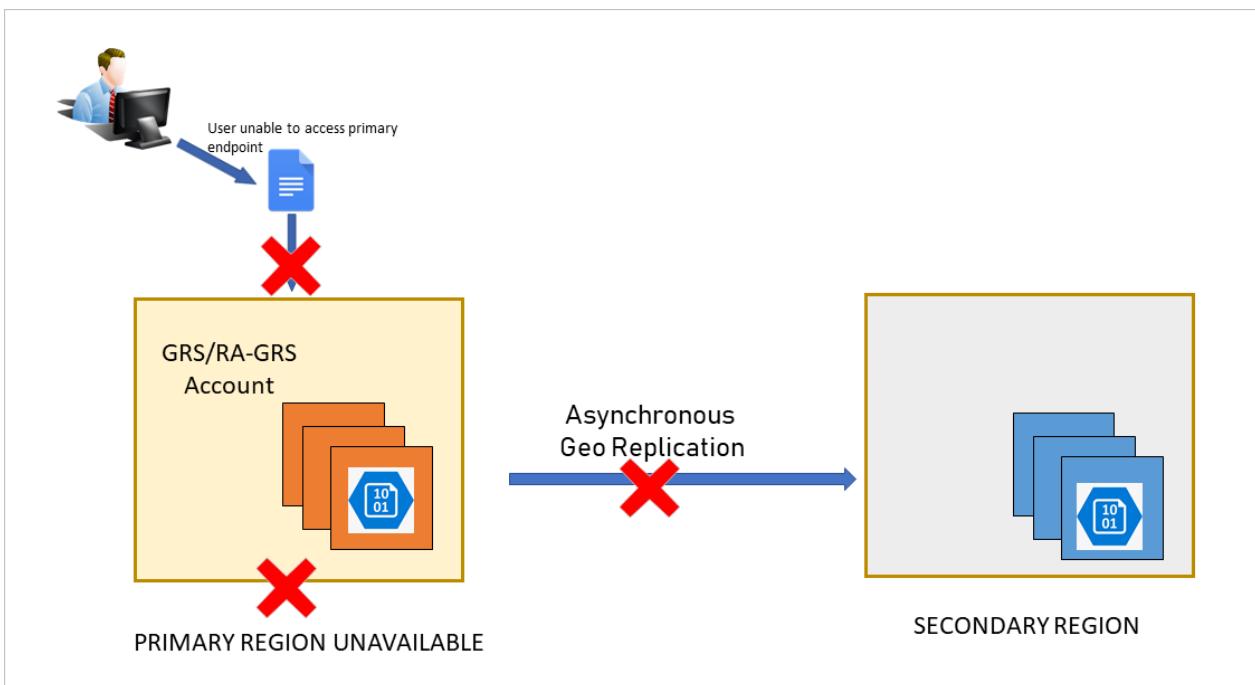
Customer-managed account failover (preview) enables you to fail your entire storage account over to the secondary region if the primary becomes unavailable for any reason. When you force a failover to the secondary region, clients can begin writing data to the secondary endpoint after the failover is complete. The failover typically takes about an hour.

### How an account failover works

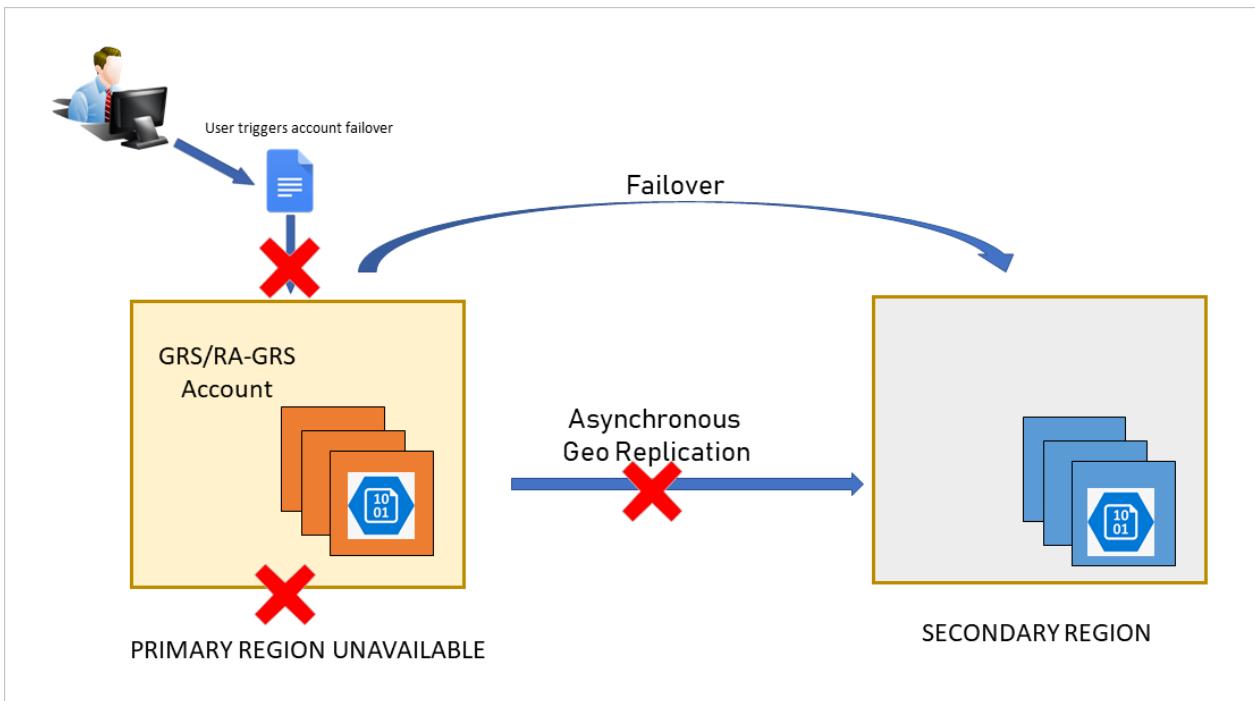
Under normal circumstances, a client writes data to an Azure Storage account in the primary region, and that data is copied asynchronously to the secondary region. The following image shows the scenario when the primary region is available:



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 GRS and RA-GRS 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 to use geo-redundant storage again (either RA-GRS or GRS).

Keep in mind that converting an LRS account to RA-GRS or GRS incurs a cost. This cost applies to updating the storage account in the new primary region to use RA-GRS or GRS after a 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 and the storage account is configured to be locally redundant. 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.

## 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 configure the account for geo-redundancy again by updating it to use GRS or RA-GRS. When the account is configured for geo-redundancy again 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 primary is fully copied to the new secondary.

After the storage account is reconfigured for geo-redundancy, it's possible to initiate another failover from the new primary back 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 locally redundant. All data in the post-failover primary region (the original secondary) is then lost. 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.

## 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 \(preview\)](#).

## About the preview

Account failover is available in preview for all customers using GRS or RA-GRS with Azure Resource Manager deployments. General-purpose v1, General-purpose v2, and Blob storage account types are supported. Account failover is currently available in all public regions. Account failover is not available in sovereign/national clouds at this time.

The preview is intended for non-production use only. Production service-level agreements (SLAs) are not currently available.

### 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 during the preview period.

#### Storage account containing archived blobs

Storage accounts containing archived blobs support account failover. Once failover is complete, to convert the account back to GRS or RA-GRS all archived blobs need to be rehydrated to an online tier first.

#### Storage resource provider

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 for the preview release:

- 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.
- ADLS Gen2 storage accounts (accounts that have hierarchical namespace enabled) 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.
- After the failover is complete, the following features may stop working if originally enabled: [Event subscriptions](#), [Change Feed](#), [Lifecycle policies](#), and [Storage Analytics Logging](#).

## Copying data as an alternative to failover

If your storage account is configured for RA-GRS, then you have read access to your data using 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.

## See also

- [Initiate an account failover \(preview\)](#)
- [Designing highly available applications using RA-GRS](#)
- [Tutorial: Build a highly available application with Blob storage](#)

# Designing highly available applications using read-access geo-redundant storage

2/12/2020 • 19 minutes to read • [Edit Online](#)

A common feature of cloud-based infrastructures like Azure Storage is that they provide a highly available platform for hosting applications. Developers of cloud-based applications must consider carefully how to leverage this platform to deliver highly available applications to their users. This article focuses on how developers can use one of Azure's geo-redundant replication options to ensure that their Azure Storage applications are highly available.

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 types of geo-redundant replication:

- [Geo-zone-redundant storage \(GZRS\) \(preview\)](#) provides replication for scenarios requiring both high availability and maximum durability. 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).
- [Geo-redundant storage \(GRS\)](#) provides cross-regional replication to protect against regional outages. 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.

For information about which primary regions are paired with which secondary regions, see [Business continuity and disaster recovery \(BCDR\): Azure Paired Regions](#).

There are code snippets included in this article, and a link to a complete sample at the end that you can download and run.

## 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 \(preview\) in Azure Storage](#).

#### **NOTE**

Customer-managed account failover (preview) is not yet available in regions supporting GZRS/RA-GZRS, so customers cannot currently manage account failover events with GZRS and RA-GZRS accounts. During the preview, Microsoft will manage any failover events affecting GZRS/RA-GZRS accounts.

## **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.

1. 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.
2. 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.
3. 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.

```
operationContext.Retrying += (sender, arguments) =>
{
 // Retrying in the primary region
 if (arguments.Request.Host == primaryhostname)
 ...
};
```

- 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.

```
public RetryInfo Evaluate(RetryContext retryContext,
 OperationContext operationContext)
{
 var statusCode = retryContext.LastRequestResult.HttpStatusCode;
 if (retryContext.CurrentRetryCount >= this.maximumAttempts
 || ((statusCode >= 300 && statusCode < 500 && statusCode != 408)
 || statusCode == 501 // Not Implemented
 || statusCode == 505 // Version Not Supported
))
 {
 // Do not retry
 return null;
 }

 // Monitor retries in the primary location
 ...

 // Determine RetryInterval and TargetLocation
 RetryInfo info =
 CreateRetryInfo(retryContext.CurrentRetryCount);

 return info;
}
```

- 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.

TIME	TRANSACTION	REPLICATION	LAST SYNC TIME	RESULT
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.
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:

```
static function OnBeforeResponse(oSession: Session) {
 ...
 if ((oSession.hostname == "\[yourstorageaccount\].table.core.windows.net")
 && (oSession.PathAndQuery.StartsWith("/employeedata?$filter")))
 {
 oSession.responseCode = 502;
 }
}
```

You could extend this example to intercept a wider range of requests and only change the `responseCode` on some of them to better simulate a real-world scenario. For more information about customizing Fiddler scripts, see [Modifying a Request or Response](#) in the Fiddler documentation.

If you have made the thresholds for switching your application to read-only mode configurable, it will be easier to test the behavior with non-production transaction volumes.

## Next Steps

- For more information about how to read from the secondary region, including another example of how the Last Sync Time property is set, see [Azure Storage Redundancy Options and Read Access Geo-Redundant Storage](#).
- 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](#).

# Azure Blob storage: hot, cool, and archive access tiers

4/15/2020 • 18 minutes to read • [Edit Online](#)

Azure storage offers different access tiers, which allow you to store blob object data in the most cost-effective manner. The available access tiers include:

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

The following considerations apply to the different access tiers:

- Only the hot and cool access tiers can be set at the account level. The archive access tier isn't available at the account level.
- Hot, cool, and archive tiers can be set at the blob level during upload or after upload.
- Data in the cool access tier can tolerate slightly lower availability, but still requires high durability, retrieval latency, and throughput characteristics similar to hot data. For cool data, a slightly lower availability service-level agreement (SLA) and higher access costs compared to hot data are acceptable trade-offs for lower storage costs.
- Archive storage stores data offline and offers the lowest storage costs but also the highest data rehydrate and access costs.

Data stored in the cloud grows at an exponential pace. To manage costs for your expanding storage needs, it's helpful to organize your data based on attributes like frequency-of-access and planned retention period to optimize costs. Data stored in the cloud can be different based on how it's generated, processed, and accessed over its lifetime. Some data is actively accessed and modified throughout its lifetime. Some data is accessed frequently early in its lifetime, with access dropping drastically as the data ages. Some data remains idle in the cloud and is rarely, if ever, accessed after it's stored.

Each of these data access scenarios benefits from a different access tier that is optimized for a particular access pattern. With hot, cool, and archive access tiers, Azure Blob storage addresses this need for differentiated access tiers with separate pricing models.

## NOTE

The features described in this article are now available to accounts that have a hierarchical namespace. To review limitations, see the [Known issues with Azure Data Lake Storage Gen2](#) article.

## Storage accounts that support tiering

Object storage data tiering between hot, cool, and archive is only supported in Blob storage and General Purpose v2 (GPv2) accounts. General Purpose v1 (GPv1) accounts don't support tiering. Customers can easily convert their existing GPv1 or Blob storage accounts to GPv2 accounts through the Azure portal. GPv2 provides new pricing and features for blobs, files, and queues. Some features and price cuts are only offered in GPv2 accounts. Evaluate using GPv2 accounts after comprehensively reviewing pricing. Some workloads can be more expensive on GPv2 than GPv1. For more information, see [Azure storage account overview](#).

Blob storage and GPv2 accounts expose the **Access Tier** attribute at the account level. This attribute allows you to specify the default access tier for any blob that doesn't have it explicitly set at the object level. For objects with the tier set at the object level, the account tier won't apply. The archive tier can be applied only at the object level. You can switch between these access tiers at any time.

## Hot access tier

The hot access tier has higher storage costs than cool and archive tiers, but the lowest access costs. Example usage scenarios for the hot access tier include:

- Data that's in active use or expected to be accessed (read from and written to) frequently.
- Data that's staged for processing and eventual migration to the cool access tier.

## Cool access tier

The cool access tier has lower storage costs and higher access costs compared to hot storage. This tier is intended for data that will remain in the cool tier for at least 30 days. Example usage scenarios for the cool access tier include:

- Short-term backup and disaster recovery datasets.
- Older media content not viewed frequently anymore but is expected to be available immediately when accessed.
- Large data sets that need to be stored cost effectively while more data is being gathered for future processing. (*For example*, long-term storage of scientific data, raw telemetry data from a manufacturing facility)

## Archive access tier

The archive access tier has the lowest storage cost. But it has higher data retrieval costs compared to the hot and cool tiers. Data in the archive tier can take several hours to retrieve. Data must remain in the archive tier for at least 180 days or be subject to an early deletion charge.

While a blob is in archive storage, the blob data is offline and can't be read, overwritten, or modified. To read or download a blob in archive, you must first rehydrate it to an online tier. You can't take snapshots of a blob in archive storage. However, the blob metadata remains online and available, allowing you to list the blob and its properties. For blobs in archive, the only valid operations are GetBlobProperties, GetBlobMetadata, ListBlobs, SetBlobTier, CopyBlob, and DeleteBlob. See [Rehydrate blob data from the archive tier](#) to learn more.

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.

## Account-level tiering

Blobs in all three access tiers can coexist within the same account. Any blob that doesn't have an explicitly assigned tier infers the tier from the account access tier setting. If the access tier comes from the account, you see the **Access Tier Inferred** blob property set to "true", and the **Access Tier** blob property matches the account tier. In the Azure portal, the *access tier inferred* property is displayed with the blob access tier as **Hot (inferred)** or **Cool (inferred)**.

Changing the account access tier applies to all *access tier inferred* objects stored in the account that don't have an explicit tier set. If you toggle the account tier from hot to cool, you'll be charged for write operations (per 10,000) for all blobs without a set tier in GPv2 accounts only. There's no charge for this change in Blob storage

accounts. You'll be charged for both read operations (per 10,000) and data retrieval (per GB) if you toggle from cool to hot in Blob storage or GPv2 accounts.

## Blob-level tiering

Blob-level tiering allows you to upload data to the access tier of your choice using the [Put Blob](#) or [Put Block List](#) operations and change the tier of your data at the object level using the [Set Blob Tier](#) operation or [Lifecycle management](#) feature. You can upload data to your required access tier then easily change the blob access tier among the hot, cool, or archive tiers as usage patterns change, without having to move data between accounts. All tier change requests happen immediately and tier changes between hot and cool are instantaneous. However, rehydrating a blob from archive can take several hours.

The time of the last blob tier change is exposed via the [Access Tier Change Time](#) blob property. When overwriting a blob in the hot or cool tier, the newly created blob inherits the tier of the blob that was overwritten unless the new blob access tier is explicitly set on creation. If a blob is in the archive tier, it can't be overwritten, so uploading the same blob isn't permitted in this scenario.

### NOTE

Archive storage and blob-level tiering only support block blobs. You also cannot currently change the tier of a block blob that has snapshots.

## Blob lifecycle management

Blob Storage lifecycle management offers a rich, rule-based policy that you can use to transition your data to the best access tier and to expire data at the end of its lifecycle. See [Manage the Azure Blob storage lifecycle](#) to learn more.

### NOTE

Data stored in a block blob storage account (Premium performance) cannot currently 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 access 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.

## Blob-level tiering billing

When a blob is uploaded or moved to the hot, cool, or archive tier, it is charged at the corresponding rate immediately upon tier change.

When a blob is moved to a cooler tier (hot->cool, hot->archive, or cool->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->cool, archive->hot, or cool->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.

[Rehydrating data from archive](#) takes time and data will be charged archive prices until the data is restored online and blob tier changes to hot or cool. The following table summarizes how tier changes are billed:

WRITE CHARGES (OPERATION + ACCESS)	READ CHARGES (OPERATION + ACCESS)
------------------------------------	-----------------------------------

	WRITE CHARGES (OPERATION + ACCESS)	READ CHARGES (OPERATION + ACCESS)
<b>SetBlobTier Direction</b>	hot->cool, hot->archive, cool->archive	archive->cool, archive->hot, cool->hot

### Cool and archive early deletion

Any blob that is moved into the cool tier (GPv2 accounts only) is subject to a cool early deletion period of 30 days. Any blob that is moved into the archive tier is subject to an archive early deletion period of 180 days. This charge is prorated. For example, if a blob is moved to archive 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 archive.

You may calculate the early deletion by using the blob property, **Last-Modified**, if there has been no access tier changes. Otherwise you can use when the access tier was last modified to cool or archive by viewing the blob property: **access-tier-change-time**. For more information on blob properties, see [Get Blob Properties](#).

## Comparing block blob storage options

The following table shows a comparison of premium performance block blob storage, and the hot, cool, and archive access tiers.

	PREMIUM PERFORMANCE	HOT TIER	COOL TIER	ARCHIVE TIER
<b>Availability</b>	99.9%	99.9%	99%	Offline
<b>Availability (RA-GRS reads)</b>	N/A	99.99%	99.9%	Offline
<b>Usage charges</b>	Higher storage costs, lower access and transaction cost	Higher storage costs, lower access, and transaction costs	Lower storage costs, higher access, and transaction costs	Lowest storage costs, highest access, and transaction costs
<b>Minimum object size</b>	N/A	N/A	N/A	N/A
<b>Minimum storage duration</b>	N/A	N/A	30 days <sup>1</sup>	180 days
<b>Latency (Time to first byte)</b>	Single-digit milliseconds	milliseconds	milliseconds	hours <sup>2</sup>

<sup>1</sup> Objects in the cool tier on GPv2 accounts have a minimum retention duration of 30 days. Blob storage accounts don't have a minimum retention duration for the cool tier.

<sup>2</sup> Archive Storage currently supports 2 rehydrate priorities, High and Standard, that offers different retrieval latencies. For more information, see [Rehydrate blob data from the archive tier](#).

#### NOTE

Blob storage accounts support the same performance and scalability targets as general-purpose v2 storage accounts. For more information, see [Scalability and performance targets for Blob storage](#).

# Quickstart scenarios

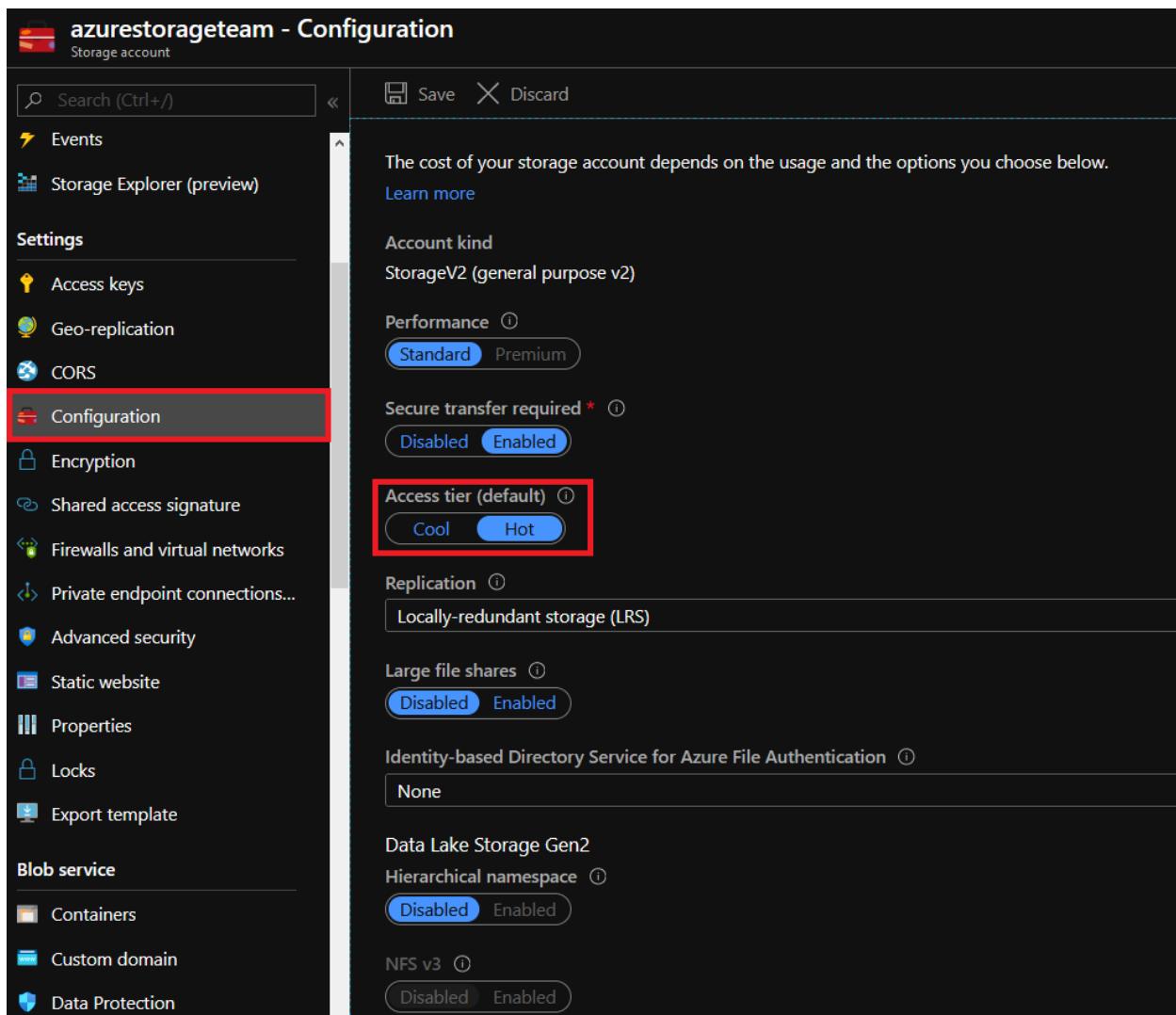
In this section, the following scenarios are demonstrated using the Azure portal and powershell:

- How to change the default account access tier of a GPv2 or Blob storage account.
- How to change the tier of a blob in a GPv2 or Blob storage account.

## Change the default account access tier of a GPv2 or Blob storage account

- [Portal](#)
- [Powershell](#)

1. Sign in to the [Azure portal](#).
2. In the Azure portal, search for and select **All Resources**.
3. Select your storage account.
4. In **Settings**, select **Configuration** to view and change the account configuration.
5. Select the right access tier for your needs: Set the **Access tier** to either Cool or Hot.
6. Click **Save** at the top.



## Change the tier of a blob in a GPv2 or Blob storage account

- [Portal](#)
- [Powershell](#)

1. Sign in to the [Azure portal](#).

2. In the Azure portal, search for and select **All Resources**.
3. Select your storage account.
4. Select your container and then select your blob.
5. In the **Blob properties**, select **Change tier**.
6. Select the **Hot**, **Cool**, or **Archive** access tier. If your blob is currently in archive and you want to rehydrate to an online tier, you may also select a Rehydrate Priority of **Standard** or **High**.
7. Select **Save** at the bottom.

The screenshot shows the Azure Storage Blob properties page for 'blob.png'. On the left, there's a toolbar with Save, Discard, Download, Refresh, Delete, Change tier (which is highlighted with a red box), Acquire lease, and Break. Below the toolbar, a note says 'This blob is currently rehydrating or archived and can't be downloaded.' The main area has tabs for Overview, Snapshots, Edit, and Generate SAS. Under Properties, there's a table with columns like URL, LAST MODIFIED, CREATION TIME, TYPE, SIZE, ACCESS TIER, etc. The ACCESS TIER is listed as 'Archive'. On the right, a 'Change tier' dialog box is open. It has sections for 'Access tier' (set to 'Hot') and 'Rehydrate priority' (set to 'High'). A note in the dialog says 'Rehydrating a blob from Archive to Hot or Cool may take several hours to complete.' At the bottom of the dialog are 'Save' and 'Cancel' buttons.

## Pricing and billing

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

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

## NOTE

For more information about pricing for Block blobs, see [Azure Storage Pricing](#) page. For more information on outbound data transfer charges, see [Data Transfers Pricing Details](#) page.

# FAQ

## Should I use Blob storage or GPv2 accounts if I want to tier my data?

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

Pricing structure between GPv1 and GPv2 accounts is different and customers should carefully evaluate both before deciding to use GPv2 accounts. You can easily convert an existing Blob storage or GPv1 account to GPv2 through a simple one-click process. For more information, see [Azure storage account overview](#).

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

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

## Can I change the default access tier of my Blob or GPv2 storage account?

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

## Can I set my default account access tier to archive?

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

## In which regions are the hot, cool, and archive access tiers available in?

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

## Do the blobs in the cool access tier behave differently than the ones in the hot access tier?

Blobs in the hot access tier have the same latency as blobs in GPv1, GPv2, and Blob storage accounts. Blobs in the cool access tier have a similar latency (in milliseconds) as blobs in GPv1, GPv2, and Blob storage accounts. Blobs in the archive access tier have several hours of latency in GPv1, GPv2, and Blob storage accounts.

Blobs in the cool access tier have a slightly lower availability service level (SLA) than the blobs stored in the hot access tier. For more information, see [SLA for storage](#).

## Are the operations among the hot, cool, and archive tiers the same?

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

operations are supported while in archive.

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

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

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

Each blob is always billed according to the tier indicated by the blob's **Access Tier** property. When you set a new online tier for a blob, the **Access Tier** property immediately reflects the new tier for all transitions.

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

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

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

## Which Azure tools and SDKs support blob-level tiering and archive storage?

Azure portal, PowerShell, and CLI tools and .NET, Java, Python, and Node.js client libraries all support blob-level tiering and archive storage.

## How much data can I store in the hot, cool, and archive tiers?

Data storage along with other limits are set at the account level and not per access tier. You can choose to use all of your limit in one tier or across all three tiers. For more information, see [Scalability and performance targets for standard storage accounts](#).

## Next steps

Evaluate hot, cool, and archive in GPv2 and Blob storage accounts

- [Check availability of hot, cool, and archive by region](#)
- [Manage the Azure Blob storage lifecycle](#)
- [Learn about rehydrating blob data from the archive tier](#)
- [Determine if premium performance would benefit your app](#)
- [Evaluate usage of your current storage accounts by enabling Azure Storage metrics](#)
- [Check hot, cool, and archive pricing in Blob storage and GPv2 accounts by region](#)
- [Check data transfers pricing](#)

# Rehydrate blob data from the archive tier

4/20/2020 • 6 minutes to read • [Edit Online](#)

While a blob is in the archive access tier, it's considered offline and can't be read or modified. The blob metadata remains online and available, allowing you to list the blob and its properties. Reading and modifying blob data is only available with online tiers such as hot or cool. There are two options to retrieve and access data stored in the archive access tier.

1. [Rehydrate an archived blob to an online tier](#) - Rehydrate an archive blob to hot or cool by changing its tier using the [Set Blob Tier](#) operation.
2. [Copy an archived blob to an online tier](#) - Create a new copy of an archive blob by using the [Copy Blob](#) operation. Specify a different blob name and a destination tier of hot or cool.

For more information on tiers, see [Azure Blob storage: hot, cool, and archive access tiers](#).

## Rehydrate an archived blob to an online tier

To read data in archive storage, you must first change the tier of the blob to hot or cool. This process is known as rehydration and can take hours to complete. We recommend large blob sizes for optimal rehydration performance. Rehydrating several small blobs concurrently may add additional time. There are currently two rehydrate priorities, High and Standard, which can be set via the optional *x-ms-rehydrate-priority* property on a [Set Blob Tier](#) or [Copy Blob](#) operation.

- **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 requests and may finish in under 1 hour. High priority may take longer than 1 hour, depending on blob size and current demand. High priority requests are guaranteed to be prioritized over Standard priority requests.

### NOTE

Standard priority is the default rehydration option for archive. High priority is a faster option that will cost more than Standard priority rehydration and is usually reserved for use in emergency data restoration situations.

Once a rehydration request is initiated, it cannot be canceled. During the rehydration process, the *x-ms-access-tier* blob property will continue to show as archive until rehydration is completed to an online tier. To confirm rehydration status and progress, you may call [Get Blob Properties](#) to check the *x-ms-archive-status* and the *x-ms-rehydrate-priority* blob properties. The archive status can read "rehydrate-pending-to-hot" or "rehydrate-pending-to-cool" depending on the rehydrate destination tier. The rehydrate priority will indicate the speed of "High" or "Standard". Upon completion, the archive status and rehydrate priority properties are removed, and the access tier blob property will update to reflect the selected hot or cool tier.

## Copy an archived blob to an online tier

If you don't want to rehydrate your archive blob, you can choose to do a [Copy Blob](#) operation. Your original blob will remain unmodified in archive while a new blob is created in the online hot or cool tier for you to work on. In the Copy Blob operation, you may also set the optional *x-ms-rehydrate-priority* property to Standard or High to specify the priority at which you want your blob copy created.

Copying a blob from archive can take hours to complete depending on the rehydrate priority selected. Behind the

scenes, the **Copy Blob** operation reads your archive source blob to create a new online blob in the selected destination tier. The new blob may be visible when you list blobs but the data is not available until the read from the source archive blob is complete and data is written to the new online destination blob. The new blob is as an independent copy and any modification or deletion to it does not affect the source archive blob.

Archive blobs can only be copied to online destination tiers within the same storage account. Copying an archive blob to another archive blob is not supported. The following table indicates CopyBlob's capabilities.

	HOT TIER SOURCE	COOL TIER SOURCE	ARCHIVE TIER SOURCE
Hot tier destination	Supported	Supported	Supported within the same account; pending rehydrate
Cool tier destination	Supported	Supported	Supported within the same account; pending rehydrate
Archive tier destination	Supported	Supported	Unsupported

## Pricing and billing

Rehydrating blobs out of archive into hot or cool tiers are charged as read operations and data retrieval. Using High priority 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 archive blob of a few gigabytes takes over 5 hours, you won't be charged the high priority retrieval rate. However, standard retrieval rates still apply as the rehydration was prioritized over other requests.

Copying blobs from archive into hot or cool tiers are charged as read operations and data retrieval. A write operation is charged for the creation of the new blob copy. 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 rehydrating archived blobs before 180 days will incur early deletion fees.

### NOTE

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](#).

## Quickstart scenarios

### Rehydrate an archive blob to an online tier

- [Portal](#)
- [PowerShell](#)

1. Sign in to the [Azure portal](#).
2. In the Azure portal, search for and select **All Resources**.
3. Select your storage account.
4. Select your container and then select your blob.
5. In the **Blob properties**, select **Change tier**.
6. Select the **Hot** or **Cool** access tier.

7. Select a Rehydrate Priority of **Standard** or **High**.

8. Select **Save** at the bottom.

The screenshot shows two Azure Storage Blob properties pages side-by-side. The top page is for 'blob.png' and the bottom page is for 'blob.txt'. Both pages have a 'Change tier' button highlighted with a red box. A modal dialog titled 'Change tier' is open over both pages. The dialog contains the following information:

- Access tier:** Hot (selected)
- Rehydrate priority:** High (selected)
- A note: 'Rehydrating a blob from Archive to Hot or Cool may take several hours to complete.'
- Access tier last modified:** 1/8/2020, 1:29:58 PM

At the bottom of the dialog are 'Save' and 'Cancel' buttons. The main blob pages also have 'Save' and 'Cancel' buttons at the bottom.

### Copy an archive blob to a new blob with an online tier

The following PowerShell script can be used to copy an archive blob to a new blob within the same storage account. The `$rgName` variable must be initialized with your resource group name. The `$accountName` variable must be initialized with your storage account name. The `$srcContainerName` and `$destContainerName` variables must be initialized with your container names. The `$srcBlobName` and `$destBlobName` variables must be initialized with your blob names.

```
#Initialize the following with your resource group, storage account, container, and blob names
$rgName = ""
$accountName = ""
$srcContainerName = ""
$destContainerName = ""
$srcBlobName = ""
$destBlobName = ""

#Select the storage account and get the context
$storageAccount =Get-AzStorageAccount -ResourceGroupName $rgName -Name $accountName
$cxt = $storageAccount.Context

#Copy source blob to a new destination blob with access tier hot using standard rehydrate priority
Start-AzStorageBlobCopy -SrcContainer $srcContainerName -SrcBlob $srcBlobName -DestContainer
$destContainerName -DestBlob $destBlobName -StandardBlobTier Hot -RehydratePriority Standard -Context $cxt
```

## Next Steps

- [Learn about Blob Storage Tiers](#)
- [Check hot, cool, and archive pricing in Blob storage and GPv2 accounts by region](#)
- [Manage the Azure Blob storage lifecycle](#)
- [Check data transfers pricing](#)

# Performance tiers for block blob storage

11/20/2019 • 3 minutes to read • [Edit Online](#)

As enterprises deploy performance sensitive cloud-native applications, it's important to have options for cost-effective data storage at different performance levels.

Azure block blob storage offers two different performance tiers:

- **Premium**: optimized for high transaction rates and single-digit consistent storage latency
- **Standard**: optimized for high capacity and high throughput

The following considerations apply to the different performance tiers:

AREA	STANDARD PERFORMANCE	PREMIUM PERFORMANCE
Region availability	All regions	In <a href="#">select regions</a>
Supported <a href="#">storage account types</a>	General purpose v2, BlobStorage, General purpose v1	BlockBlobStorage
Supports <a href="#">high throughput block blobs</a>	Yes, at greater than 4 MiB PutBlock or PutBlob sizes	Yes, at greater than 256 KiB PutBlock or PutBlob sizes
Redundancy	See <a href="#">Types of storage accounts</a>	Currently supports only locally-redundant storage (LRS) and zone-redundant storage (ZRS) <sup>1</sup>

<sup>1</sup>Zone-redundant storage (ZRS) is available in select regions for premium performance block blob storage accounts.

Regarding cost, premium performance provides optimized pricing for applications with high transaction rates to help [lower total storage cost](#) for these workloads.

## Premium performance

Premium performance block blob storage 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.

Premium performance storage is ideal for workloads that require fast and consistent response times. It's best for workloads that perform many small transactions. Example workloads include:

- **Interactive workloads**. These workloads require instant updates and user feedback, such as e-commerce and mapping applications. 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.
- **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.

- **Data transformation.** Processes that require constant editing, modification, and conversion of data require instant updates. For accurate data representation, consumers of this data must see these changes reflected immediately.

## Standard performance

Standard performance supports different [access tiers](#) to store data in the most cost-effective manner. It's optimized for high capacity and high throughput on large data sets.

- **Backup and disaster recovery datasets.** Standard performance storage offers cost-efficient tiers, making it a perfect use case for both short-term and long-term disaster recovery datasets, secondary backups, and compliance data archiving.
- **Media content.** Images and videos often are accessed frequently when first created and stored, but this content type is used less often as it gets older. Standard performance storage offers suitable tiers for media content needs.
- **Bulk data processing.** These kinds of workloads are suitable for standard storage because they require cost-effective high-throughput storage instead of consistent low latency. Large, raw datasets are staged for processing and eventually migrate to cooler tiers.

## Migrate from standard to premium

You can't convert an existing standard performance storage account to a block blob storage account with premium performance. To migrate to a premium performance storage account, you must create a BlockBlobStorage account, and migrate the data to the new account. For more information, see [Create a BlockBlobStorage account](#).

To copy blobs between storage accounts, you can use the latest version of the [AzCopy](#) command-line tool. Other tools such as Azure Data Factory are also available for data movement and transformation.

## Blob lifecycle management

Blob storage lifecycle management offers a rich, rule-based policy:

- **Premium:** Expire data at the end of its lifecycle.
- **Standard:** Transition data to the best access tier and expire data at the end of its lifecycle.

To learn more, see [Manage the Azure Blob storage lifecycle](#).

You can't move data that's stored in a premium block blob storage account between hot, cool, and archive tiers. However, you can copy blobs from a block blob storage account to the hot access tier in a *different* account. To copy data to a different account, use the [Put Block From URL API](#) or [AzCopy v10](#). The **Put Block From URL API** synchronously copies data on the server. The call completes only after all the data is moved from the original server location to the destination location.

## Next steps

Evaluate hot, cool, and archive in GPv2 and Blob storage accounts.

- [Learn about rehydrating blob data from the archive tier](#)
- [Evaluate usage of your current storage accounts by enabling Azure Storage metrics](#)
- [Check hot, cool, and archive pricing in Blob storage and GPv2 accounts by region](#)
- [Check data transfers pricing](#)

# Manage the Azure Blob storage lifecycle

2/25/2020 • 11 minutes to read • [Edit Online](#)

Data sets have unique lifecycles. Early in the lifecycle, people access some data often. But the need for access drops drastically as the data ages. Some data stays idle in the cloud and is rarely accessed once stored. Some data expires days or months after creation, while other data sets are actively read and modified throughout their lifetimes. Azure Blob storage lifecycle management offers a rich, rule-based policy for GPv2 and Blob storage accounts. Use the policy to transition your data to the appropriate access tiers or expire at the end of the data's lifecycle.

The lifecycle management policy lets you:

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

Consider a scenario where data gets frequent access 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 adjusting storage tiers in respect to the age of data, you can design the least expensive storage options for your needs. To achieve this transition, lifecycle management policy rules are available to move aging data to cooler tiers.

## NOTE

The features described in this article are now available to accounts that have a hierarchical namespace. To review limitations, see the [Known issues with Azure Data Lake Storage Gen2](#) article.

## Storage account support

The lifecycle management policy is available with General Purpose v2 (GPv2) accounts, Blob storage accounts, and Premium Block Blob storage accounts. In the Azure portal, you can upgrade an existing General Purpose (GPv1) account to a GPv2 account. For more information about storage accounts, see [Azure storage account overview](#).

## Pricing

The lifecycle management feature is free of charge. Customers are charged the regular operation cost for the [List Blobs](#) and [Set Blob Tier](#) API calls. Delete operation is free. For more information about pricing, see [Block Blob pricing](#).

## Regional availability

The lifecycle management feature is available in all Azure regions.

## Add or remove a policy

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

- [Azure portal](#)
- [Azure PowerShell](#)
- [Azure CLI](#)
- [REST APIs](#)

A policy can be read or written in full. Partial updates are not supported.

**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](#).

This article shows how to manage policy by using the portal and PowerShell methods.

- [Portal](#)
- [Powershell](#)
- [Template](#)

There are two ways to add a policy through the Azure portal.

- [Azure portal List view](#)
- [Azure portal Code view](#)

**Azure portal List view**

1. Sign in to the [Azure portal](#).
2. In the Azure portal, search for and select your storage account.
3. Under **Blob Service**, select **Lifecycle management** to view or change your rules.
4. Select the **List view** tab.
5. Select **Add rule** and then fill out the **Action set** form fields. In the following example, blobs are moved to cool storage if they haven't been modified for 30 days.

The screenshot shows the 'Add a rule' wizard in the Microsoft Azure portal. The current step is 'Action set'. A red box highlights the 'Action set' tab. The page content describes how each rule definition includes an action set and a filter set. It lists actions for blobs: 'Move blob to cool storage' (checked, with 'Days after last modification' set to 30), 'Move blob to archive storage' (unchecked), 'Delete blob' (unchecked). It also lists actions for snapshots: 'Delete snapshot' (unchecked, with 'Days after blob is created' set to 0).

6. Select **Filter set** to add an optional filter. Then, select **Browse** to specify a container and folder by which to filter.

The screenshot shows the 'Add a rule' wizard in the Microsoft Azure portal. The current step is 'Filter set'. A red box highlights the 'Filter set' tab. The page content describes how each rule definition includes an action set and a filter set. It lists supported blob types: 'Block Blob'. Under 'Prefix match', it says to apply a rule to a container or a subset of virtual folders with up to 10 prefixes. A 'Browse' button is highlighted with a red box. A 'Browse' dialog is open on the right, showing a list of containers: 'cs4316e81020662x41cbxb95'. It includes instructions about prefix matching and a dropdown for filtering containers by prefix. At the bottom of the dialog are 'Select' and 'Cancel' buttons.

7. Select **Review + add** to review the policy settings.
  8. Select **Add** to add the new policy.
- Azure portal Code view**
1. Sign in to the [Azure portal](#).
  2. In the Azure portal, search for and select your storage account.
  3. Under **Blob Service**, select **Lifecycle management** to view or change your policy.
  4. The following JSON is an example of a policy that can be pasted into the **Code view** tab.

```
{
 "rules": [
 {
 "name": "ruleFoo",
 "enabled": true,
 "type": "Lifecycle",
 "definition": {
 "filters": {
 "blobTypes": ["blockBlob"],
 "prefixMatch": ["container1/foo"]
 },
 "actions": {
 "baseBlob": {
 "tierToCool": { "daysAfterModificationGreaterThan": 30 },
 "tierToArchive": { "daysAfterModificationGreaterThan": 90 },
 "delete": { "daysAfterModificationGreaterThan": 2555 }
 },
 "snapshot": {
 "delete": { "daysAfterCreationGreaterThan": 90 }
 }
 }
 }
 }
]
}
```

5. Select **Save**.
6. For more information about this JSON example, see the [Policy](#) and [Rules](#) sections.

## Policy

A lifecycle management policy is a collection of rules in a JSON document:

```
{
 "rules": [
 {
 "name": "rule1",
 "enabled": true,
 "type": "Lifecycle",
 "definition": {...}
 },
 {
 "name": "rule2",
 "type": "Lifecycle",
 "definition": {...}
 }
]
}
```

A policy is a collection of rules:

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:

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 temporary 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

## Rules

Each rule definition includes a filter set and an action set. The [filter set](#) limits rule actions to a certain set of objects within a container or objects names. The [action set](#) applies the tier or delete actions to the filtered set of objects.

### Sample rule

The following sample rule filters the account to run the actions on objects that exist inside `container1` and start with `foo`.

#### NOTE

Lifecycle management only supports block blob type.

- 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 blob snapshots 90 days after snapshot creation

```
{
 "rules": [
 {
 "name": "ruleFoo",
 "enabled": true,
 "type": "Lifecycle",
 "definition": {
 "filters": {
 "blobTypes": ["blockBlob"],
 "prefixMatch": ["container1/foo"]
 },
 "actions": {
 "baseBlob": {
 "tierToCool": { "daysAfterModificationGreaterThan": 30 },
 "tierToArchive": { "daysAfterModificationGreaterThan": 90 },
 "delete": { "daysAfterModificationGreaterThan": 2555 }
 },
 "snapshot": {
 "delete": { "daysAfterCreationGreaterThan": 90 }
 }
 }
 }
 }
]
}
```

## 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 <code>blockBlob</code> .	Yes
prefixMatch	An array of strings for prefixes to be match. Each rule can define up to 10 prefixes. A prefix string must start with a container name. For example, if you want to match all blobs under <a href="https://myaccount.blob.core.windows.net/container1/foo/">https://myaccount.blob.core.windows.net/container1/foo/</a> for a rule, the prefixMatch is <code>container1/foo</code> .	If you don't define prefixMatch, the rule applies to all blobs within the storage account.	No

## Rule actions

Actions are applied to the filtered blobs when the run condition is met.

Lifecycle management supports tiering and deletion of blobs and deletion of blob snapshots. Define at least one action for each rule on blobs or blob snapshots.

ACTION	BASE BLOB	SNAPSHOT
tierToCool	Support blobs currently at hot tier	Not supported

Action	Base Blob	Snapshot
tierToArchive	Support blobs currently at hot or cool tier	Not supported
delete	Supported	Supported

#### 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. Base blobs use the last modified time to track age, and blob snapshots use the snapshot creation time to track age.

Action Run Condition	Condition Value	Description
<code>daysAfterModificationGreaterThan</code>	Integer value indicating the age in days	The condition for base blob actions
<code>daysAfterCreationGreaterThan</code>	Integer value indicating the age in days	The condition for blob snapshot actions

## Examples

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 `container1/foo` or `container2/bar`. 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": ["container1/foo", "container2/bar"]
 },
 "actions": {
 "baseBlob": {
 "tierToCool": { "daysAfterModificationGreaterThan": 30 },
 "tierToArchive": { "daysAfterModificationGreaterThan": 90 }
 }
 }
 }
 }
]
}
```

### Archive data after ingest

Some data stays idle in the cloud and is rarely, if ever, accessed once stored. The following lifecycle policy is

configured to archive data shortly after it is ingested. This example transitions block blobs in the storage account within container `archivecontainer` into an archive tier. The transition is accomplished by acting on blobs 0 days after last modified time:

#### NOTE

It is recommended to upload your blobs directly to the archive tier to be more efficient. You can use the `x-ms-access-tier` header for [PutBlob](#) or [PutBlockList](#) with REST version 2018-11-09 and newer or our latest blob storage client libraries.

```
{
 "rules": [
 {
 "name": "archiveRule",
 "enabled": true,
 "type": "Lifecycle",
 "definition": {
 "filters": {
 "blobTypes": ["blockBlob"],
 "prefixMatch": ["archivecontainer"]
 },
 "actions": {
 "baseBlob": {
 "tierToArchive": { "daysAfterModificationGreaterThan": 0 }
 }
 }
 }
 }
]
}
```

#### 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 older than 365 days.

```
{
 "rules": [
 {
 "name": "expirationRule",
 "enabled": true,
 "type": "Lifecycle",
 "definition": {
 "filters": {
 "blobTypes": ["blockBlob"]
 },
 "actions": {
 "baseBlob": {
 "delete": { "daysAfterModificationGreaterThan": 365 }
 }
 }
 }
 }
]
}
```

#### Delete old snapshots

For data that is modified and accessed regularly throughout its lifetime, snapshots are often used to track older versions of the data. You can create a policy that deletes old snapshots based on snapshot age. The snapshot age

is determined by evaluating the snapshot creation time. This policy rule deletes block blob snapshots within container `activedata` that are 90 days or older after snapshot creation.

```
{
 "rules": [
 {
 "name": "snapshotRule",
 "enabled": true,
 "type": "Lifecycle",
 "definition": {
 "filters": {
 "blobTypes": ["blockBlob"],
 "prefixMatch": ["activedata"]
 },
 "actions": {
 "snapshot": {
 "delete": { "daysAfterCreationGreaterThanOrEqual": 90 }
 }
 }
 }
 }
]
}
```

## 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.

**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.

## Next steps

Learn how to recover data after accidental deletion:

- [Soft delete for Azure Storage blobs](#)

# Performance and scalability checklist for Blob storage

3/31/2020 • 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>
	Caching	<a href="#">Is your application caching data that is frequently accessed and rarely changed?</a>

DONE	CATEGORY	DESIGN CONSIDERATION
	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?
	Uploading quickly	When trying to upload many blobs quickly, are you uploading blobs in parallel?

DONE	CATEGORY	DESIGN CONSIDERATION
	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 role-based access control (RBAC) roles on a per-container basis. For more information, see [Grant access to Azure blob and queue data with RBAC in the Azure portal](#).
- 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 4 MiB for standard storage accounts and greater than 256 KiB for 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 deliver 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. For more information, see the [Azure Storage reference documentation](#).

## 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 (exchangable 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

1/8/2020 • 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 [Azure Blob storage: hot, cool, and archive access tiers](#).

## 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.

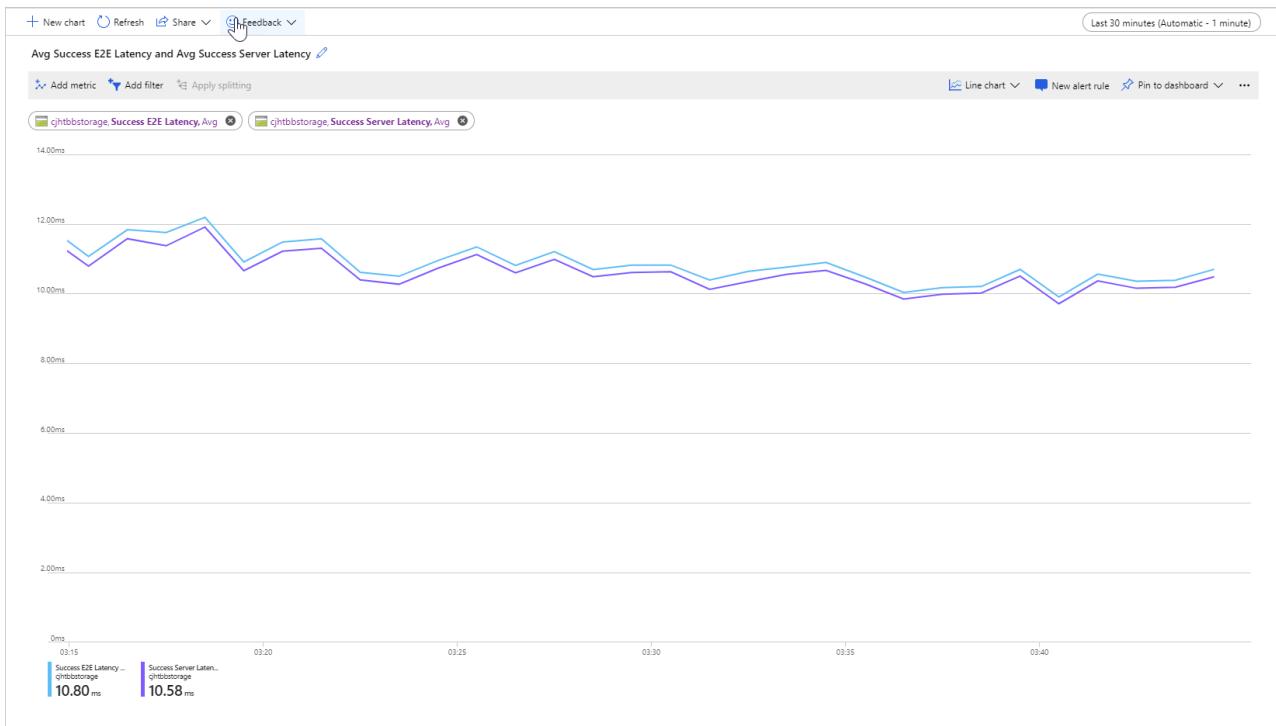
Azure Storage bandwidth, also referred to as throughput, is related to the request rate and can be calculated by multiplying the request rate (IOPS) by the request size. For example, assuming 160 requests per second, each 256 KiB of data results in throughput of 40,960 KiB per second or 40 MiB per second.

## 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.

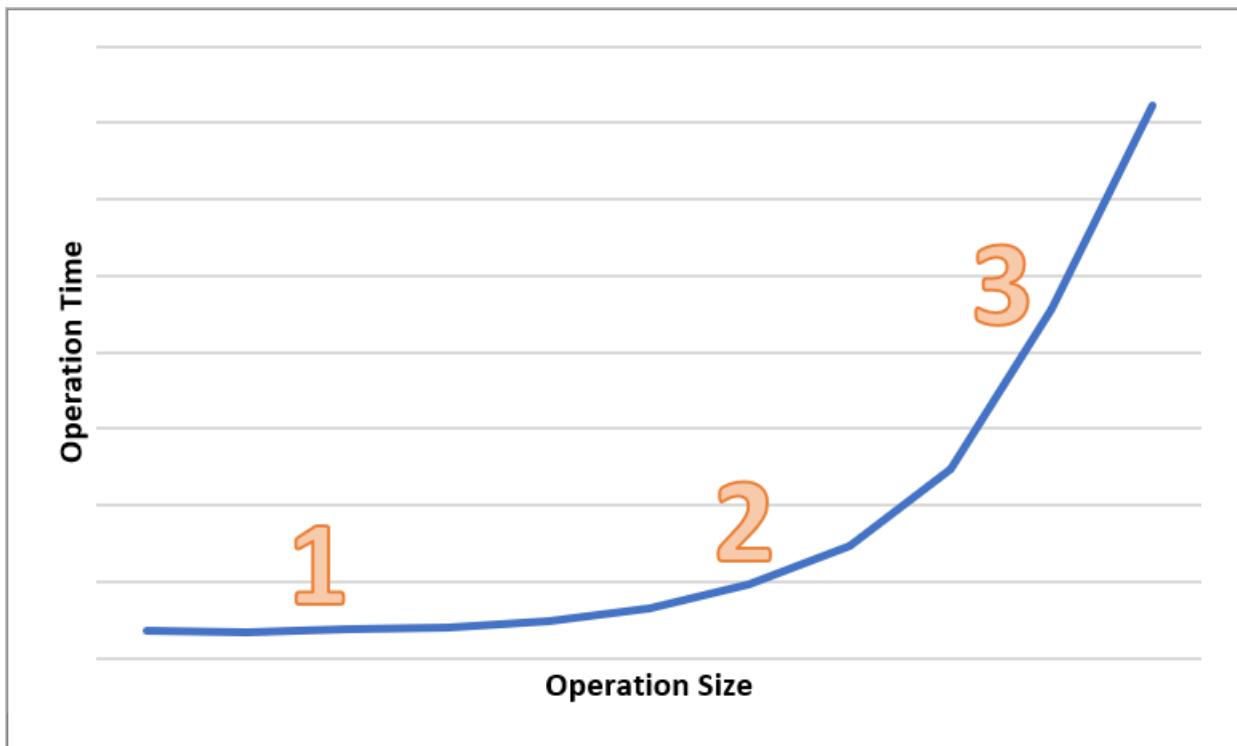
If you review your end-to-end and server latency metrics, and find that end-to-end latency is significantly higher than server latency, then investigate and address the source of the additional latency.

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

3/27/2020 • 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 TB and 1 PB per month for a one-year or three-year term.

### Reservation scope

Azure Storage reserved capacity is available for a single subscription or for multiple subscriptions (shared scope). 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 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. A reservation cannot be split across multiple subscriptions.

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 [Azure Blob storage: hot, cool, and archive access tiers](#).

All types of redundancy are supported for reservations. For more information about redundancy options, see [Azure Storage redundancy](#).

#### **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, Azure Table storage, or Azure Files.

## **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 TB or 1 PB blocks, with higher discounts for 1 PB 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 \*  Subscription \*

Region : **West US** Access tier : **Hot** Redundancy : **ZRS** Size : **Select a value**

Billing frequency : **Monthly**

↑↓ 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

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 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>
Region	The region where the reservation is in effect.

FIELD	DESCRIPTION
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="#">Azure Blob storage: hot, cool, and archive access tiers</a> .
Redundancy	The redundancy option for the reservation. Options include <i>LRS</i> , <i>ZRS</i> , <i>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 region where the reservation is in effect.
Term	One year or three years.

- 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.
- 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

Next: Review + buy

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, minus a 12 percent early termination fee. 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 Storage

3/4/2020 • 3 minutes to read • [Edit Online](#)

This article describes how you plan and manage costs for Azure Storage. First, you use the Azure pricing calculator to help plan for storage costs before you add any resources. After you begin using Azure Storage resources, 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 Azure Storage are only a portion of the monthly costs in your Azure bill. Although this article explains how to plan for and manage costs for Azure 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 Azure Storage, you can apply similar methods to manage costs for all the Azure services used in your subscription.

## Prerequisites

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](#).

## Estimate costs before creating an Azure Storage account

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, there's a header with 'Your Estimate' and navigation buttons. Below it is a summary box with 'Your Estimate' and 'Estimate total: \$21.94'. To the right are icons for edit, delete, and clone, along with a link to '\$21.94'. The main area is titled 'Storage Accounts' and contains a table with dropdown menus for configuration. The table columns are: REGION, TYPE, PERFORMANCE TIER, STORAGE ACCOUNT TYPE, REDUNDANCY, and ACCESS TIER. The REGION is set to 'West US', TYPE is 'Block Blob Storage', PERFORMANCE TIER is 'Standard', STORAGE ACCOUNT TYPE is 'Blob Storage', REDUNDANCY is 'LRS', and ACCESS TIER is 'Hot'. In the bottom right corner of the calculator interface, there is a red-bordered 'More info' box containing links to 'Pricing details', 'Product details', and 'Documentation'.

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.

# Use budgets and cost alerts

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.

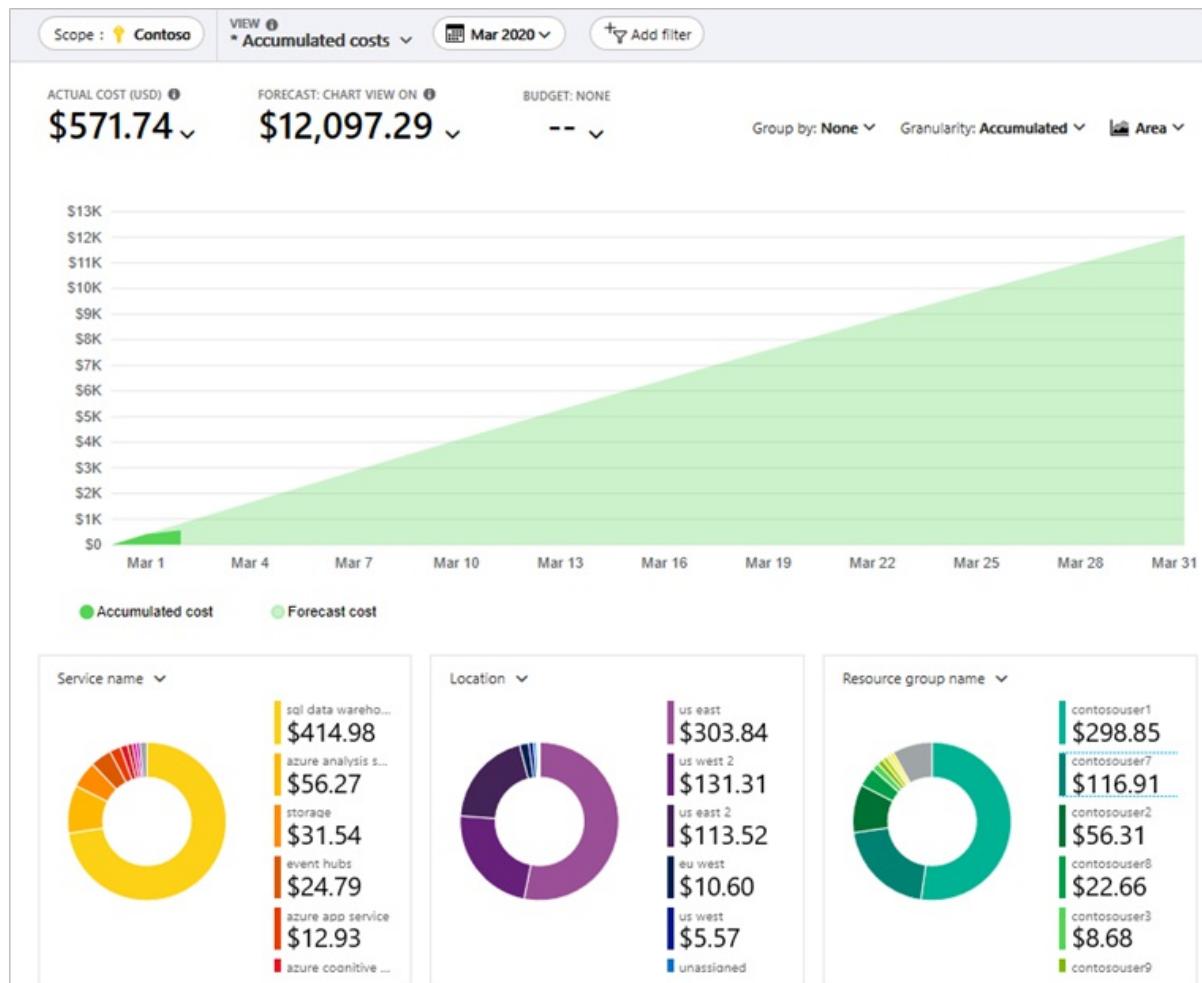
## Monitor costs

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.

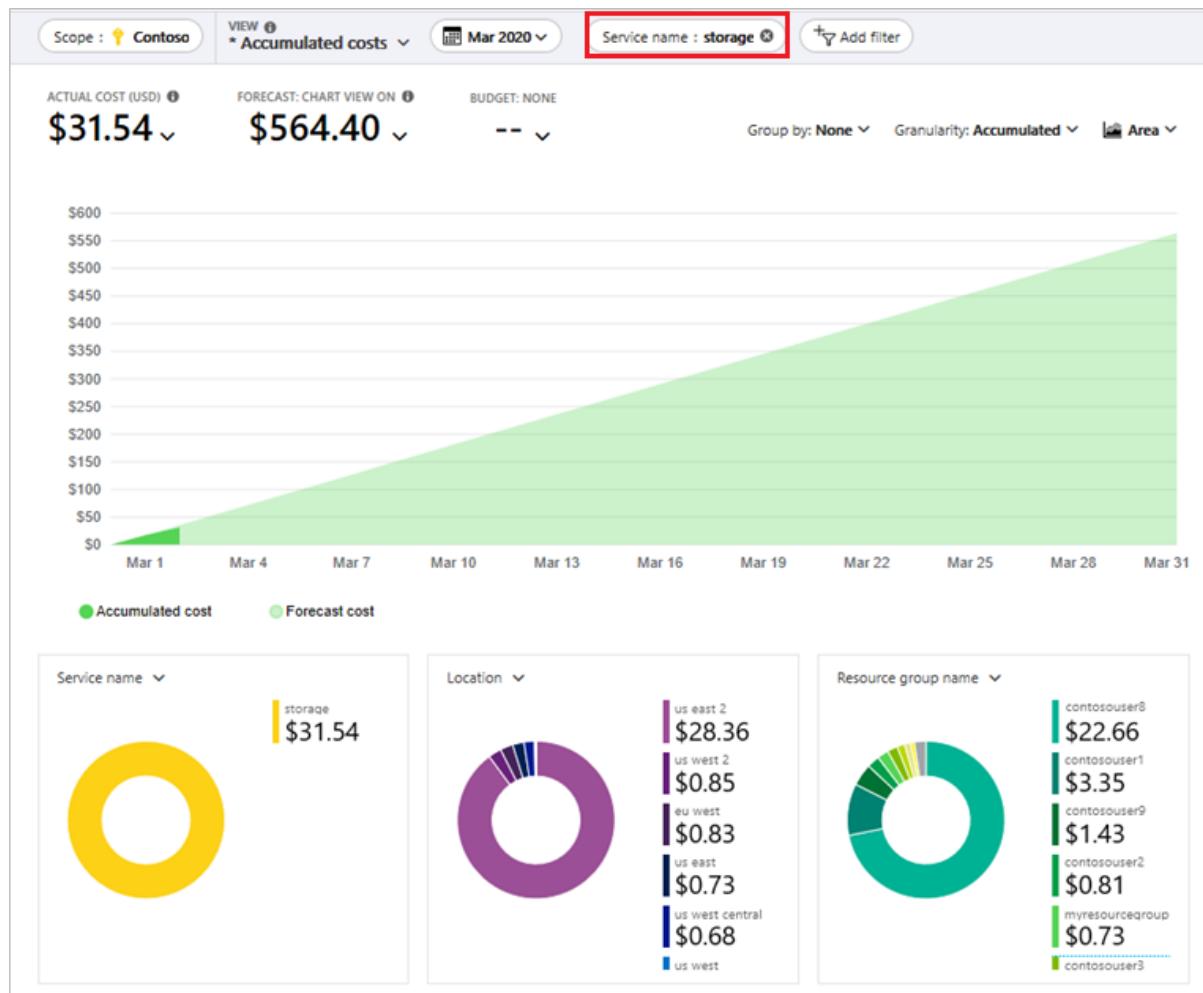
To view Azure Storage costs in cost analysis:

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.

## Next steps

Learn more about managing costs with [cost analysis](#).

See the following articles to learn more on how pricing works with Azure Storage:

- [Azure Storage Overview pricing](#)
- [Optimize costs for Blob storage with reserved capacity](#)

# Managing Concurrency in Microsoft Azure Storage

12/23/2019 • 16 minutes to read • [Edit Online](#)

Modern Internet-based applications typically have multiple users viewing and updating data simultaneously. This requires application developers 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:

1. Optimistic concurrency – An application performing an update will as part of its update verify if the data has changed since the application last read that data. For example, if two users viewing a wiki page make an update to the same page then the wiki platform must ensure that the second update does not overwrite the first update – and that both users understand whether their update was successful or not. This strategy is most often used in web applications.
2. 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 master/subordinate data replication scenario where only the master will perform updates the master will typically hold an exclusive lock for an extended period of time on the data to ensure no one else can update it.
3. Last writer wins – An approach that allows any update operations to proceed without verifying if any other application has updated the data since the application first read the data. This strategy (or lack of a formal strategy) is usually used where data is partitioned in such a way that there is no likelihood that multiple users will access the same data. It can also be useful where short-lived data streams are being processed.

This article provides an overview of how the Azure Storage platform simplifies development by providing first class support for all three of these concurrency strategies.

## Azure Storage simplifies cloud development

The Azure storage service supports all three strategies, although it is distinctive in its ability to provide full support for optimistic and pessimistic concurrency because it was designed to embrace a strong consistency model which guarantees that when the Storage service commits a data insert or update operation all further accesses to that data will see the latest update. Storage platforms that use an eventual consistency model have a lag between when a write is performed by one user and when the updated data can be seen by other users thus complicating development of client applications in order to prevent inconsistencies from affecting end users.

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. The Azure storage service uses snapshot isolation to allow read operations to happen concurrently with write operations within a single partition. Unlike other isolation levels, snapshot isolation guarantees that all reads see a consistent snapshot of the data even while updates are occurring – essentially by returning the last committed values while an update transaction is being processed.

## Managing concurrency in Blob storage

You can opt to use either optimistic or pessimistic concurrency models to manage access to blobs and containers in the Blob service. If you do not explicitly specify a strategy last writes wins is the default.

### Optimistic concurrency for blobs and containers

The Storage service assigns an identifier to every object stored. This identifier is updated every time an update operation is performed on an object. The identifier is returned to the client as part of an HTTP GET response using the ETag (entity tag) header that is defined within the HTTP protocol. A user performing an update on such an object

can send in the original ETag along with a conditional header to ensure that an update will only occur if a certain condition has been met – in this case the condition is an "If-Match" header, which requires the Storage Service to ensure the value of the ETag specified in the update request is the same as that stored in the Storage Service.

The outline of this process is as follows:

1. Retrieve a blob from the storage service, the response includes an HTTP ETag Header value that identifies the current version of the object in the storage service.
2. When you update the blob, include the ETag value you received in step 1 in the **If-Match** conditional header of the request you send to the service.
3. The service compares the ETag value in the request with the current ETag value of the blob.
4. If the current ETag value of the blob is a different version than the ETag in the **If-Match** conditional header in the request, the service returns a 412 error to the client. This indicates to the client that another process has updated the blob since the client retrieved it.
5. If the current ETag value of the blob is the same version as the ETag in the **If-Match** conditional header in the request, the service performs the requested operation and updates the current ETag value of the blob to show that it has created a new version.

The following C# snippet (using the Client Storage Library 4.2.0) shows a simple example of how to construct an **If-Match AccessCondition** based on the ETag value that is accessed from the properties of a blob that was previously either retrieved or inserted. It then uses the **AccessCondition** object when it updates the blob: the **AccessCondition** object adds the **If-Match** header to the request. If another process has updated the blob, the Blob service returns an HTTP 412 (Precondition Failed) status message. You can download the full sample here: [Managing Concurrency using Azure Storage](#).

```
// Retrieve the ETag from the newly created blob
// Etag is already populated as UploadText should cause a PUT Blob call
// to storage Blob service which returns the ETag in response.
string originalETag = blockBlob.Properties.ETag;

// This code simulates an update by a third party.
string helloText = "Blob updated by a third party.';

// No ETag provided so original blob is overwritten (thus generating a new ETag)
blockBlob.UploadText(helloText);
Console.WriteLine("Blob updated. Updated ETag = {0}",
blockBlob.Properties.ETag);

// Now try to update the blob using the original ETag provided when the blob was created
try
{
 Console.WriteLine("Trying to update blob using original ETag to generate if-match access condition");
 blockBlob.UploadText(helloText,accessCondition:
 AccessCondition.GenerateIfMatchCondition(originalETag));
}
catch (StorageException ex)
{
 if (ex.RequestInformation.HttpStatusCode == (int)HttpStatusCode.PreconditionFailed)
 {
 Console.WriteLine("Precondition failure as expected. Blob's original ETag no longer matches");
 // TODO: client can decide on how it wants to handle the 3rd party updated content.
 }
 else
 throw;
}
```

Azure Storage also includes support for additional conditional headers such as **If-Modified-Since**, **If-Unmodified-Since** and **If-None-Match** as well as combinations thereof. For more information, see [Specifying Conditional Headers for Blob Service Operations](#).

The following table summarizes the container operations that accept conditional headers such as **If-Match** in the request and that return an ETag value in the response.

OPERATION	RETURNS CONTAINER ETAG VALUE	ACCEPTS CONDITIONAL HEADERS
Create Container	Yes	No
Get Container Properties	Yes	No
Get Container Metadata	Yes	No
Set Container Metadata	Yes	Yes
Get Container ACL	Yes	No
Set Container ACL	Yes	Yes (*)
Delete Container	No	Yes
Lease Container	Yes	Yes
List Blobs	No	No

(\*) The permissions defined by SetContainerACL are cached and updates to these permissions take 30 seconds to propagate during which period updates are not guaranteed to be consistent.

The following table summarizes the blob operations that accept conditional headers such as **If-Match** in the request and that return an ETag value in the response.

OPERATION	RETURNS ETAG VALUE	ACCEPTS CONDITIONAL HEADERS
Put Blob	Yes	Yes
Get Blob	Yes	Yes
Get Blob Properties	Yes	Yes
Set Blob Properties	Yes	Yes
Get Blob Metadata	Yes	Yes
Set Blob Metadata	Yes	Yes
Lease Blob (*)	Yes	Yes
Snapshot Blob	Yes	Yes
Copy Blob	Yes	Yes (for source and destination blob)
Abort Copy Blob	No	No
Delete Blob	No	Yes

OPERATION	RETURNS ETAG VALUE	ACCEPTS CONDITIONAL HEADERS
Put Block	No	No
Put Block List	Yes	Yes
Get Block List	Yes	No
Put Page	Yes	Yes
Get Page Ranges	Yes	Yes

(\*) Lease Blob does not change the ETag on a blob.

### Pessimistic concurrency for blobs

To lock a blob for exclusive use, you can acquire a [lease](#) on it. When you acquire a lease, you specify for how long you need the lease: this can be for between 15 to 60 seconds or infinite, which amounts to an exclusive lock. You can renew a finite lease to extend it, and you can release any lease when you are finished with it. The Blob service automatically releases finite leases when they expire.

Leases enable different synchronization strategies to be supported, including exclusive write / shared read, exclusive write / exclusive read and shared write / exclusive read. Where a lease exists the storage service enforces exclusive writes (put, set and delete operations) however ensuring exclusivity for read operations requires the developer to ensure 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 C# snippet shows an example of acquiring an exclusive lease for 30 seconds on a blob, updating the content of the blob, and then releasing the lease. If there is already a valid lease on the blob when you try to acquire a new lease, the Blob service returns an "HTTP (409) Conflict" status result. The following snippet uses an **AccessCondition** object to encapsulate the lease information when it makes a request to update the blob in the storage service. You can download the full sample here: [Managing Concurrency using Azure Storage](#).

```
// Acquire lease for 15 seconds
string lease = blockBlob.AcquireLease(TimeSpan.FromSeconds(15), null);
Console.WriteLine("Blob lease acquired. Lease = {0}", lease);

// Update blob using lease. This operation will succeed
const string helloText = "Blob updated";
var accessCondition = AccessCondition.GenerateLeaseCondition(lease);
blockBlob.UploadText(helloText, accessCondition: accessCondition);
Console.WriteLine("Blob updated using an exclusive lease");

//Simulate third party update to blob without lease
try
{
 // Below operation will fail as no valid lease provided
 Console.WriteLine("Trying to update blob without valid lease");
 blockBlob.UploadText("Update without lease, will fail");
}
catch (StorageException ex)
{
 if (ex.RequestInformation.HttpStatusCode == (int)HttpStatusCode.PreconditionFailed)
 Console.WriteLine("Precondition failure as expected. Blob's lease does not match");
 else
 throw;
}
```

If you attempt a write operation on a leased blob without passing the lease ID, the request fails with a 412 error.

Note that if the lease expires before calling the [UploadText](#) method but you still pass the lease ID, the request also fails with a 412 error. For more information about managing lease expiry times and lease IDs, see the [Lease Blob](#) REST documentation.

The following blob operations can use leases to manage pessimistic concurrency:

- Put Blob
- Get Blob
- Get Blob Properties
- Set Blob Properties
- Get Blob Metadata
- Set Blob Metadata
- Delete Blob
- Put Block
- Put Block List
- Get Block List
- Put Page
- Get Page Ranges
- Snapshot Blob - lease ID optional if a lease exists
- Copy Blob - lease ID required if a lease exists on the destination blob
- Abort Copy Blob - lease ID required if an infinite lease exists on the destination blob
- Lease Blob

### Pessimistic concurrency for containers

Leases on containers enable the same synchronization strategies to be supported as on blobs (exclusive write / shared read, exclusive write / exclusive read and shared write / exclusive read) however unlike blobs the storage service only enforces exclusivity 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 succeed on a leased container without including the lease ID in which case they are shared operations. If exclusivity of update (put or set) or read operations is required then developers should ensure all clients use a lease ID and that only one client at a time has a valid lease ID.

The following container operations can use leases to manage pessimistic concurrency:

- Delete Container
- Get Container Properties
- Get Container Metadata
- Set Container Metadata
- Get Container ACL
- Set Container ACL
- Lease Container

For more information, see:

- [Specifying Conditional Headers for Blob Service Operations](#)
- [Lease Container](#)
- [Lease Blob](#)

## Managing concurrency in Table storage

The Table service uses optimistic concurrency checks as the default behavior when you are working with entities, unlike the Blob service where you must explicitly choose to perform optimistic concurrency checks. The other

difference between the table and Blob services is that you can only manage the concurrency behavior of entities whereas with the Blob service you can manage the concurrency of both containers and blobs.

To use optimistic concurrency and to check if another process modified an entity since you retrieved it from the table storage service, you can use the ETag value you receive when the table service returns an entity. The outline of this process is as follows:

1. Retrieve an entity from the table storage service, the response includes an ETag value that identifies the current identifier associated with that entity in the storage service.
2. When you update the entity, include the ETag value you received in step 1 in the mandatory **If-Match** header of the request you send to the service.
3. The service compares the ETag value in the request with the current ETag value of the entity.
4. If the current ETag value of the entity is different than the ETag in the mandatory **If-Match** header in the request, the service returns a 412 error to the client. This indicates to the client that another process has updated the entity since the client retrieved it.
5. If the current ETag value of the entity is the same as the ETag in the mandatory **If-Match** header in the request or the **If-Match** header contains the wildcard character (\*), the service performs the requested operation and updates the current ETag value of the entity to show that it has been updated.

Note that unlike the Blob service, the table service requires the client to include an **If-Match** header in update requests. However, it is possible to force an unconditional update (last writer wins strategy) and bypass concurrency checks if the client sets the **If-Match** header to the wildcard character (\*) in the request.

The following C# snippet shows a customer entity that was previously either created or retrieved having their email address updated. The initial insert or retrieve operation stores the ETag value in the customer object, and because the sample uses the same object instance when it executes the replace operation, it automatically sends the ETag value back to the table service, enabling the service to check for concurrency violations. If another process has updated the entity in table storage, the service returns an HTTP 412 (Precondition Failed) status message. You can download the full sample here: [Managing Concurrency using Azure Storage](#).

```
try
{
 customer.Email = "updatedEmail@contoso.org";
 TableOperation replaceCustomer = TableOperation.Replace(customer);
 customerTable.Execute(replaceCustomer);
 Console.WriteLine("Replace operation succeeded.");
}
catch (StorageException ex)
{
 if (ex.RequestInformation.HttpStatusCode == 412)
 Console.WriteLine("Optimistic concurrency violation - entity has changed since it was retrieved.");
 else
 throw;
}
```

To explicitly disable the concurrency check, you should set the **ETag** property of the **employee** object to "\*" before you execute the replace operation.

```
customer.ETag = "*";
```

The following table summarizes how the table entity operations use ETag values:

OPERATION	RETURNS ETAG VALUE	REQUIRES IF-MATCH REQUEST HEADER
Query Entities	Yes	No

OPERATION	RETURNS ETAG VALUE	REQUIRES IF-MATCH REQUEST HEADER
Insert Entity	Yes	No
Update Entity	Yes	Yes
Merge Entity	Yes	Yes
Delete Entity	No	Yes
Insert or Replace Entity	Yes	No
Insert or Merge Entity	Yes	No

Note that the **Insert or Replace Entity** and **Insert or Merge Entity** operations do *not* perform any concurrency checks because they do not send an ETag value to the table service.

In general developers using tables should rely on optimistic concurrency when developing scalable applications. If pessimistic locking is needed, one approach developers can take when accessing Tables is to assign a designated blob for each table and try to take a lease on the blob before operating on the table. This approach does require the application to ensure all data access paths obtain the lease prior to operating on the table. You should also note that the minimum lease time is 15 seconds which requires careful consideration for scalability.

For more information, see:

- [Operations on Entities](#)

## Managing Concurrency in the Queue Service

One scenario in which concurrency is a concern in the queueing service is where multiple clients are retrieving messages from a queue. When a message is retrieved from the queue, the response includes the message and a pop receipt value, which is required to delete the message. The message is not automatically deleted from the queue, but after it has been retrieved, it is not visible to other clients for the time interval specified by the visibilitytimeout parameter. The client that retrieves the message is expected to delete the message after it has been processed, and before the time specified by the TimeNextVisible element of the response, which is calculated based on the value of the visibilitytimeout parameter. The value of visibilitytimeout is added to the time at which the message is retrieved to determine the value of TimeNextVisible.

The queue service does not have support for either optimistic or pessimistic concurrency and for this reason clients processing messages retrieved from a queue should ensure messages are processed in an idempotent manner. A last writer wins strategy is used for update operations such as SetQueueServiceProperties, SetQueueMetaData, SetQueueACL and UpdateMessage.

For more information, see:

- [Queue Service REST API](#)
- [Get Messages](#)

## Managing concurrency in Azure Files

The file service can be accessed using two different protocol endpoints – SMB and REST. The REST service does not have support for either optimistic locking or pessimistic locking and all updates will follow a last writer wins strategy. SMB clients that mount file shares can leverage file system locking mechanisms to manage access to shared files – including the ability to perform pessimistic locking. When an SMB client opens a file, it specifies both the file access and share mode. Setting a File Access option of "Write" or "Read/Write" along with a File Share mode

of "None" will result in the file being locked by an SMB client until the file is closed. If REST operation is attempted on a file where an SMB client has the file locked the REST service will return status code 409 (Conflict) with error code SharingViolation.

When an SMB client opens a file for delete, it marks the file as pending delete until all other SMB client open handles on that file are closed. While a file is marked as pending delete, any REST operation on that file will return status code 409 (Conflict) with error code SMBDeletePending. Status code 404 (Not Found) is not returned since it is possible for the SMB client to remove the pending deletion flag prior to closing the file. In other words, status code 404 (Not Found) is only expected when the file has been removed. Note that while a file is in an SMB pending delete state, it will not be included in the List Files results. Also, note that the REST Delete File and REST Delete Directory operations are committed atomically and do not result in a pending delete state.

For more information, see:

- [Managing File Locks](#)

## Next steps

For the complete sample application referenced in this blog:

- [Managing Concurrency using Azure Storage - Sample Application](#)

For more information on Azure Storage see:

- [Microsoft Azure Storage Home Page](#)
- [Introduction to Azure Storage](#)
- Storage Getting Started for [Blob](#), [Table](#), [Queues](#), and [Files](#)
- Storage Architecture – [Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency](#)

# Use AI to understand Blob storage data

11/4/2019 • 8 minutes to read • [Edit Online](#)

Data in Azure Blob storage is often a variety of unstructured content such as images, long text, PDFs, and Office documents. By using the AI capabilities in Azure Cognitive Search, you can understand and extract valuable information from blobs in a variety of ways. Examples of applying AI to blob content include:

- Extract text from images using optical character recognition (OCR)
- Produce a scene description or tags from a photo
- Detect language and translate text into different languages
- Process text with named entity recognition (NER) to find references to people, dates, places, or organizations

While you might need just one of these AI capabilities, it's common to combine multiple of them into the same pipeline (for example, extracting text from a scanned image and then finding all the dates and places referenced in it).

AI enrichment creates new information, captured as text, stored in fields. Post-enrichment, you can access this information from a search index through full text search, or send enriched documents back to Azure storage to power new application experiences that include exploring data for discovery or analytics scenarios.

In this article, we view AI enrichment through a wide lens so that you can quickly grasp the entire process, from transforming raw data in blobs, to queryable information in either a search index or a knowledge store.

## What it means to "enrich" blob data with AI

*AI enrichment* is part of the indexing architecture of Azure Cognitive Search that integrates built-in AI from Microsoft or custom AI that you provide. It helps you implement end-to-end scenarios where you need to process blobs (both existing ones and new ones as they come in or are updated), crack open all file formats to extract images and text, extract the desired information using various AI capabilities, and index them in a search index for fast search, retrieval and exploration.

Inputs are your blobs, in a single container, in Azure Blob storage. Blobs can be almost any kind of text or image data.

Output is always a search index, used for fast text search, retrieval, and exploration in client applications. Additionally, output can also be a *knowledge store* that projects enriched documents into Azure blobs or Azure tables for downstream analysis in tools like Power BI or in data science workloads.

In between is the pipeline architecture itself. The pipeline is based on the *indexer* feature, to which you can assign a *skillset*, which is composed of one or more *skills* providing the AI. The purpose of the pipeline is to produce *enriched documents* that enter as raw content but pick up additional structure, context, and information while moving through the pipeline. Enriched documents are consumed during indexing to create inverted indexes and other structures used in full text search or exploration and analytics.

## Start with services

You need Azure Cognitive Search and Azure Blob storage. Within Blob storage, you need a container that provides source content.

You can start directly in your Storage account portal page. In the left navigation page, under **Blob service** click **Add Azure Cognitive Search** to create a new service or select an existing one.

Once you add Azure Cognitive Search to your storage account, you can follow the standard process to enrich data in any Azure data source. We recommend the **Import data** wizard in Azure Cognitive Search for an easy initial introduction to AI enrichment. This quickstart walks you through the steps: [Create an AI enrichment pipeline in the portal](#).

In the following sections, we'll explore more components and concepts.

## Use a Blob indexer

AI enrichment is an add-on to an indexing pipeline, and in Azure Cognitive Search, those pipelines are built on top of an *indexer*. An indexer is a data-source-aware subservice equipped with internal logic for sampling data, reading metadata data, retrieving data, and serializing data from native formats into JSON documents for subsequent import. Indexers are often used by themselves for import, separate from AI, but if you want to build an AI enrichment pipeline, you will need an indexer and a skillset to go with it. This section highlights the indexer; the next section focuses on skillsets.

Blobs in Azure Storage are indexed using the [Azure Cognitive Search Blob storage indexer](#). You can invoke this indexer by using 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 does the "document cracking", 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, image, and other content types are detected. Document cracking with text extraction is no charge. Document cracking with image extraction is charged at rates you can find on the [pricing page](#).

Although all documents will be cracked, enrichment only occurs if you explicitly provide the skills to do so. For example, if your pipeline consists exclusively of image analysis, text in your container or documents is ignored.

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 [Azure Cognitive Search Blob storage indexer](#).

## Add AI components

AI enrichment refers to modules that look for patterns or characteristics, and then perform an operation accordingly. Facial recognition in photos, text descriptions of photos, detecting key phrases in a document, and OCR (or recognizing printed or handwritten text in binary files) are illustrative examples.

In Azure Cognitive Search, *skills* are the individual components of AI processing that you can use standalone, or in combination with other skills.

- Built-in skills are backed by Cognitive Services, with image analysis based on Computer Vision, and natural language processing based on Text Analytics. For the complete list, see [Built-in skills for content enrichment](#).
- Custom skills are custom code, wrapped in an [interface definition](#) that allows for integration into the pipeline. In customer solutions, it's common practice to use both, with custom skills providing open-source, third-party, or first-party AI modules.

A *skillset* is the collection of skills used in a pipeline, and it's invoked after the document cracking phase makes content available. An indexer can consume exactly one skillset, but that skillset exists independently of an indexer so that you can reuse it in other scenarios.

Custom skills might sound complex but can be simple and straightforward in terms of implementation. If you have existing packages that provide pattern matching or classification models, the content you extract from blobs could be passed to these models for processing. Since AI enrichment is Azure-based, your model should be on Azure.

also. Some common hosting methodologies include using [Azure Functions](#) or [Containers](#).

Built-in skills backed by Cognitive Services require an [attached Cognitive Services](#) all-in-one subscription key that gives you access to the resource. An all-in-one key gives you image analysis, language detection, text translation, and text analytics. Other built-in skills are features of Azure Cognitive Search and require no additional service or key. Text shaper, splitter, and merger are examples of helper skills that are sometimes necessary when designing the pipeline.

If you use only custom skills and built-in utility skills, there is no dependency or costs related to Cognitive Services.

## Consume AI-enriched output in downstream solutions

The output of AI enrichment is either a search index on Azure Cognitive Search, or a [knowledge store](#) in Azure Storage.

In Azure Cognitive Search, a search index is used for interactive exploration using free text and filtered queries in a client app. Enriched documents created through AI are formatted in JSON and indexed in the same way all documents are indexed in Azure Cognitive Search, leveraging all of the benefits an indexer provides. For example, during indexing, the blob indexer refers to configuration parameters and settings to utilize any field mappings or change detection logic. Such settings are fully available to regular indexing and AI enriched workloads. Post-indexing, when content is stored on Azure Cognitive Search, you can build rich queries and filter expressions to understand your content.

In Azure Storage, a knowledge store has two manifestations: a blob container, or tables in Table storage.

- A blob container captures enriched documents in their entirety, which is useful if you want to feed into other processes.
- In contrast, Table storage can accommodate physical projections of enriched documents. You can create slices or layers of enriched documents that include or exclude specific parts. For analysis in Power BI, the tables in Azure Table storage become the data source for further visualization and exploration.

An enriched document at the end of the pipeline differs from its original input version by the presence of additional fields containing new information that was extracted or generated during enrichment. As such, you can work with a combination of original and created content, regardless of which output structure you use.

## Next steps

There's a lot more you can do with AI enrichment to get the most out of your data in Azure Storage, including combining Cognitive Services in different ways, and authoring custom skills for cases where there's no existing Cognitive Service for the scenario. You can learn more by following the links below.

- [Upload, download, and list blobs with the Azure portal \(Azure Blob storage\)](#)
- [Set up a blob indexer \(Azure Cognitive Search\)](#)
- [AI enrichment overview \(Azure Cognitive Search\)](#)
- [Create a skillset \(Azure Cognitive Search\)](#)
- [Map nodes in an annotation tree \(Azure Cognitive Search\)](#)

# Add full text search to Azure blob data using Azure Cognitive Search

11/4/2019 • 5 minutes to read • [Edit Online](#)

Searching across the variety of content types stored in Azure Blob storage can be a difficult problem to solve. However, you can index and search the content of your Blobs in just a few clicks by using [Azure Cognitive Search](#). Azure Cognitive Search has built-in integration for indexing out of Blob storage through a [\*Blob indexer\*](#) that adds data-source-aware capabilities to indexing.

## What it means to add full text search to blob data

Azure Cognitive Search is a cloud search service that provides indexing and query engines that operate over user-defined indexes hosted on your search service. Co-locating your searchable content with the query engine in the cloud is necessary for performance, returning results at a speed users have come to expect from search queries.

Azure 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, access to blob content can leverage the full range of query features in Azure Cognitive Search.

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 with changes to the underlying container. 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.

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 blob indexing](#) 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.

## Start with services

You need Azure Cognitive Search and Azure Blob storage. Within Blob storage, you need a container that provides source content.

You can start directly in your Storage account portal page. In the left navigation page, under **Blob service** click [Add Azure Cognitive Search](#) to create a new service or select an existing one.

Once you add Azure Cognitive Search to your storage account, you can follow the standard process to index blob data. We recommend the **Import data** wizard in Azure Cognitive Search for an easy initial introduction, or call the REST APIs using a tool like Postman. This tutorial walks you through the steps of calling the REST API in Postman: [Index and search semi-structured data \(JSON blobs\) in Azure Cognitive Search](#).

## Use a Blob indexer

An *indexer* is a data-source-aware subservice 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 [Azure Cognitive Search Blob storage indexer](#). You can invoke this

indexer by using 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 does the "document cracking", 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 to text content only.

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 [Azure Cognitive Search Blob storage indexer](#).

## 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:

- PDF
- 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
- HTML
- XML
- ZIP
- GZ
- EPUB
- EML
- RTF
- Plain text files (see also [Indexing plain text](#))
- JSON (see [Indexing JSON blobs](#))
- CSV (see [Indexing CSV blobs](#))

## 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.

## Indexing JSON blobs

Indexers can be configured to extract structured content found in blobs that contain JSON. An indexer can read JSON blobs and parse the structured content into the appropriate fields of a search document. Indexers can also take blobs that contain an array of JSON objects and map each element to a separate search document. You can set a parsing mode to affect the type of JSON object created by the indexer.

# Search blob content in a search index

The output of an indexing 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. You can use [simple query syntax](#), [full query syntax](#), and [filter expression syntax](#) in Search explorer.

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 \(Azure Cognitive Search\)](#)

# Choose an Azure solution for data transfer

3/4/2020 • 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

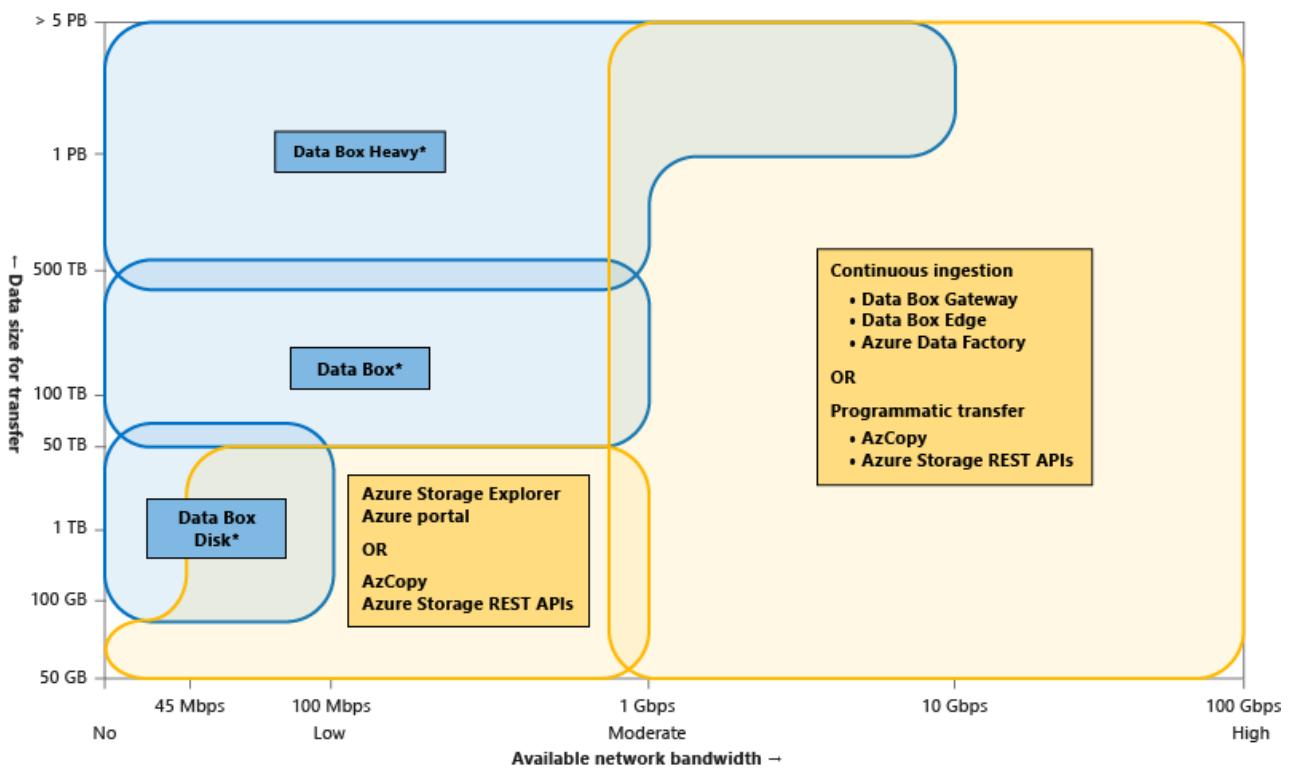
Data transfer can be offline or over the network connection. Choose your solution depending on your:

- **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 Data Box 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.](#)
- [Use Azure PowerShell with Azure Storage](#)
- [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 Data Box 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

4/12/2020 • 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>	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

## 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
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
Hardware	Microsoft supplied	Microsoft supplied	Microsoft supplied	Customer supplied
Network interface	USB 3.1/SATA	RJ 45, SFP+	RJ45, QSFP+	SATA II/SATA III
Partner integration	Some	High	High	Some
Shipping	Microsoft managed	Microsoft managed	Microsoft managed	Customer managed
Use when data moves	Within a commerce boundary	Within a commerce boundary	Within a commerce boundary	Across geographic boundaries, e.g. US to EU
Pricing	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

4/12/2020 • 4 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** – Data Box Edge and Data Box Gateway are online network devices that can move data into and out of Azure. Use Data Box 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
Hardware	Microsoft supplied	Microsoft supplied	Microsoft supplied	Customer supplied

	DATA BOX DISK	DATA BOX	DATA BOX HEAVY	IMPORT/EXPORT
Network interface	USB 3.1/SATA	RJ 45, SFP+	RJ45, QSFP+	SATA II/SATA III
Partner integration	Some	High	High	Some
Shipping	Microsoft managed	Microsoft managed	Microsoft managed	Customer managed
Use when data moves	Within a commerce boundary	Within a commerce boundary	Within a commerce boundary	Across geographic boundaries, e.g. US to EU
Pricing	<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 DATA BOX EDGE	AZURE DATA FACTORY
Data type	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
Form factor	Command-line tools	Programmatic interface	Microsoft supplies a virtual or physical device	Service in Azure portal
Initial one-time setup	Easy	Moderate	Easy (<30 minutes) to moderate (1-2 hours)	Extensive
Data pre-processing	No	No	Yes (With Edge compute)	Yes
Transfer from other clouds	No	No	No	Yes
User type	IT Pro or dev	Dev	IT Pro	IT Pro
Pricing	Free, data egress charges apply	Free, data egress charges apply	<a href="#">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.](#)
- 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.](#)
- Use the REST APIs to transfer data
  - [In .NET](#)
  - [In Java](#)

# Data transfer for small datasets with low to moderate network bandwidth

3/15/2019 • 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

6/25/2019 • 2 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 Data Box online transfer device or Azure Data Factory. These tools are set up by IT professionals and can transparently automate data transfer.
  - **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.

## 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

CAPABILITY	AZCOPY	AZURE STORAGE REST APIs
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	<a href="#">Pricing</a>	<a href="#">Pricing</a>	<a href="#">Pricing</a>

## 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.](#)

# What is Azure Import/Export service?

3/25/2020 • 8 minutes to read • [Edit Online](#)

Azure Import/Export service is used to securely import large amounts of data to Azure Blob storage and Azure Files by shipping disk drives to an Azure datacenter. This service can also be used to transfer data from Azure Blob storage to disk drives and ship to your on-premises sites. Data from one or more disk drives can be imported either to Azure Blob storage or Azure Files.

Supply your own disk drives and transfer data with the Azure Import/Export service. You can also use disk drives supplied by Microsoft.

If you want to transfer data using disk drives supplied by Microsoft, you can use [Azure Data Box Disk](#) to import data into Azure. Microsoft ships up to 5 encrypted solid-state disk drives (SSDs) with a 40 TB total capacity per order, to your datacenter through a regional carrier. You can quickly configure disk drives, copy data to disk drives over a USB 3.0 connection, and ship the disk drives back to Azure. For more information, go to [Azure Data Box Disk overview](#).

## Azure Import/Export use cases

Consider using Azure Import/Export service when uploading or downloading data over the network is too slow, or getting additional network bandwidth is cost-prohibitive. Use this service in the following scenarios:

- **Data migration to the cloud:** Move large amounts of data to Azure quickly and cost effectively.
- **Content distribution:** Quickly send data to your customer sites.
- **Backup:** Take backups of your on-premises data to store in Azure Storage.
- **Data recovery:** Recover large amount of data stored in storage and have it delivered to your on-premises location.

## Import/Export components

Import/Export service uses the following components:

- **Import/Export service:** This service available in Azure portal helps the user create and track data import (upload) and export (download) jobs.
- **WAImpoerExport tool:** This is a command-line tool that does the following:
  - Prepares your disk drives that are shipped for import.
  - Facilitates copying your data to the drive.
  - Encrypts the data on the drive with AES 128-bit BitLocker. You can use an external key protector to protect your BitLocker key.
  - Generates the drive journal files used during import creation.
  - Helps identify numbers of drives needed for export jobs.

**NOTE**

The WAImportExport tool is available in two versions, version 1 and 2. We recommend that you use:

- Version 1 for import/export into Azure Blob storage.
- Version 2 for importing data into Azure files.

The WAImportExport tool is only compatible with 64-bit Windows operating system. For specific OS versions supported, go to [Azure Import/Export requirements](#).

- **Disk Drives:** You can ship Solid-state drives (SSDs) or Hard disk drives (HDDs) to the Azure datacenter. When creating an import job, you ship disk drives containing your data. When creating an export job, you ship empty drives to the Azure datacenter. For specific disk types, go to [Supported disk types](#).

## How does Import/Export work?

Azure Import/Export service allows data transfer into Azure Blobs and Azure Files by creating jobs. Use Azure portal or Azure Resource Manager REST API to create jobs. Each job is associated with a single storage account.

The jobs can be import or export jobs. An import job allows you to import data into Azure Blobs or Azure files whereas the export job allows data to be exported from Azure Blobs. For an import job, you ship drives containing your data. When you create an export job, you ship empty drives to an Azure datacenter. In each case, you can ship up to 10 disk drives per job.

### Inside an import job

At a high level, an import job involves the following steps:

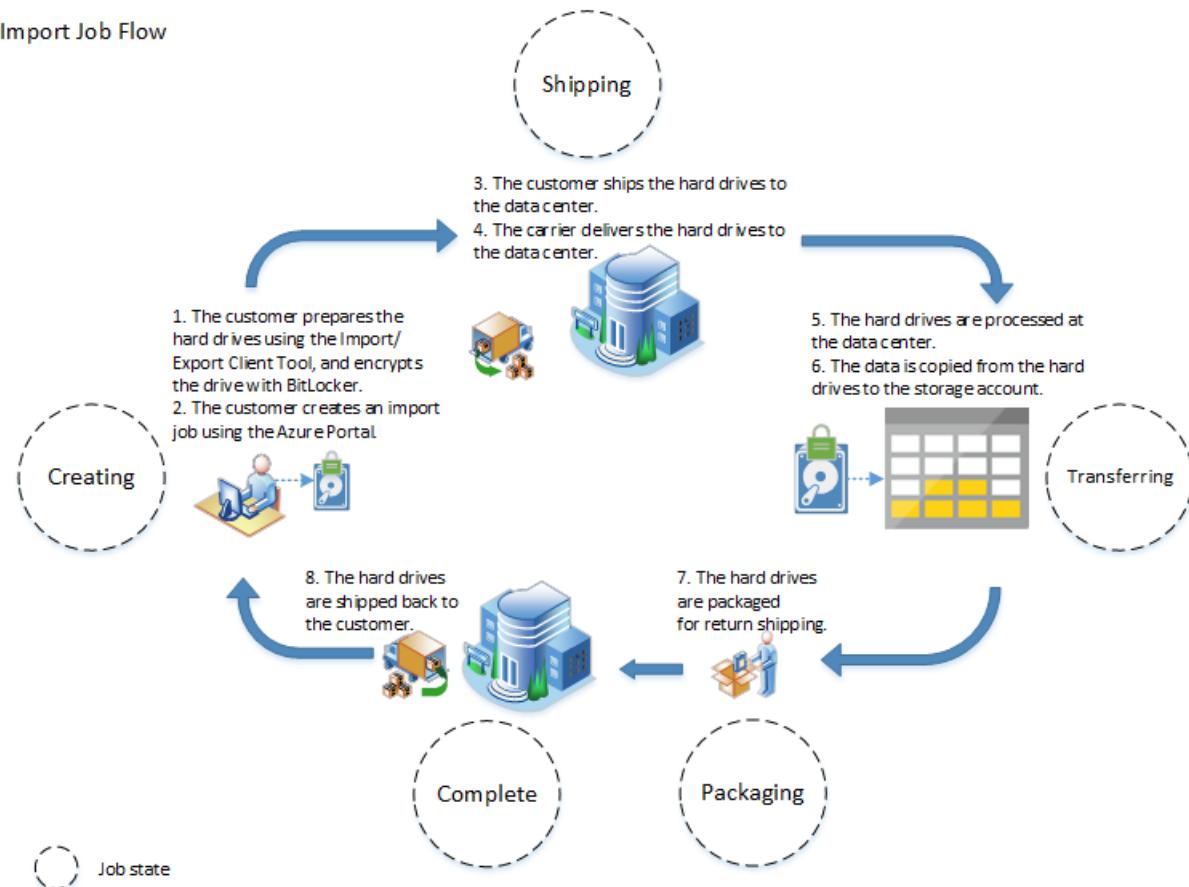
1. Determine data to be imported, number of drives you need, destination blob location for your data in Azure storage.
2. Use the WAImportExport tool to copy data to disk drives. Encrypt the disk drives with BitLocker.
3. Create an import job in your target storage account in Azure portal. Upload the drive journal files.
4. Provide the return address and carrier account number for shipping the drives back to you.
5. Ship the disk drives to the shipping address provided during job creation.
6. Update the delivery tracking number in the import job details and submit the import job.
7. The drives are received and processed at the Azure data center.
8. The drives are shipped using your carrier account to the return address provided in the import job.

**NOTE**

For local (within data center country/region) shipments, please share a domestic carrier account.

For abroad (outside data center country/region) shipments, please share an international carrier account.

## Import Job Flow



For step-by-step instructions on data import, go to:

- [Import data into Azure Blobs](#)
- [Import data into Azure Files](#)

### Inside an export job

#### IMPORTANT

The service only supports export of Azure Blobs. Export of Azure files is not supported.

At a high level, an export job involves the following steps:

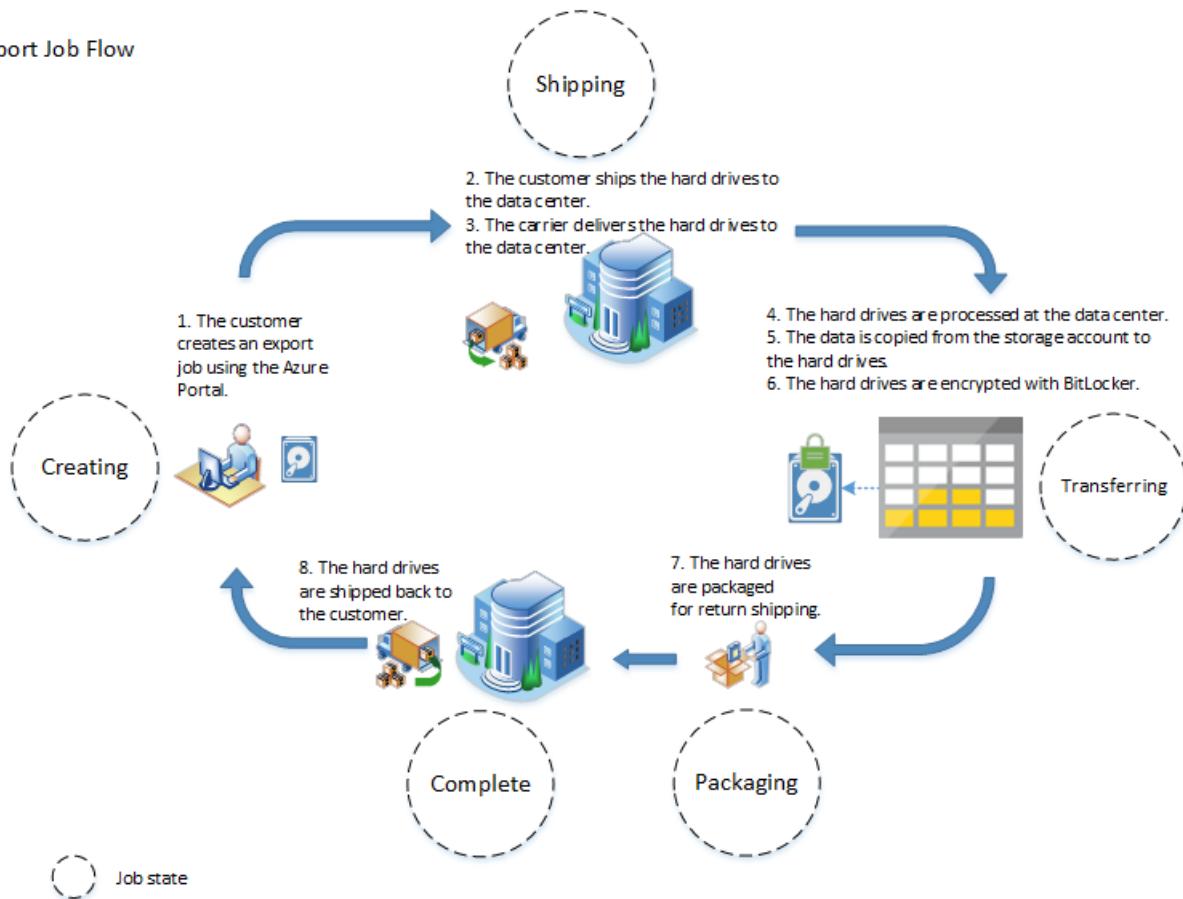
1. Determine the data to be exported, number of drives you need, source blobs or container paths of your data in Blob storage.
2. Create an export job in your source storage account in Azure portal.
3. Specify source blobs or container paths for the data to be exported.
4. Provide the return address and carrier account number for shipping the drives back to you.
5. Ship the disk drives to the shipping address provided during job creation.
6. Update the delivery tracking number in the export job details and submit the export job.
7. The drives are received and processed at the Azure data center.
8. The drives are encrypted with BitLocker and the keys are available via the Azure portal.
9. The drives are shipped using your carrier account to the return address provided in the import job.

#### NOTE

For local (within data center country/region) shipments, please share a domestic carrier account.

For abroad (outside data center country/region) shipments, please share an international carrier account.

## Export Job Flow



For step-by-step instructions on data export, go to [Export data from Azure Blobs](#).

## Region availability

The Azure Import/Export service supports copying data to and from all Azure storage accounts. You can ship disk drives to one of the listed locations. If your storage account is in an Azure location that is not specified here, an alternate shipping location is provided when you create the job.

### Supported shipping locations

COUNTRY/REGION	COUNTRY/REGION	COUNTRY/REGION	COUNTRY/REGION
East US	North Europe	Central India	US Gov Iowa
West US	West Europe	South India	US DoD East
East US 2	East Asia	West India	US DoD Central
West US 2	Southeast Asia	Canada Central	China East
Central US	Australia East	Canada East	China North
North Central US	Australia Southeast	Brazil South	UK South
South Central US	Japan West	Korea Central	Germany Central
West Central US	Japan East	US Gov Virginia	Germany Northeast

## Security considerations

The data on the drive is encrypted using AES 128-bit BitLocker Drive Encryption. This encryption protects your data while it is in transit.

For import jobs, drives are encrypted in two ways.

- Specify the option when using *dataset.csv* file while running the WAImportExport tool during drive preparation.
- Enable BitLocker encryption manually on the drive. Specify the encryption key in the *driveset.csv* when running WAImportExport tool command line during drive preparation. The BitLocker encryption key can be further protected by using an external key protector (also known as the Microsoft managed key) or a customer managed key. For more information, see how to [Use a customer managed key to protect your BitLocker key](#).

For export jobs, after your data is copied to the drives, the service encrypts the drive using BitLocker before shipping it back to you. The encryption key is provided to you via the Azure portal. The drive needs to be unlocked using the WAImportExport tool using the key.

## Deleting personal information

### NOTE

This article provides steps for how to delete personal data from the device or service and can be used to support your obligations under the GDPR. If you're looking for general info about GDPR, see the [GDPR section of the Service Trust portal](#).

Personal information is relevant to the import/export service (via the portal and API) during import and export operations. Data used during these processes include:

- Contact name
- Phone number
- Email
- Street address
- City
- Zip/postal code
- State
- Country/Province/Region
- Drive ID
- Carrier account number
- Shipping tracking number

When an import/export job is created, users provide contact information and a shipping address. Personal information is stored in up to two different locations: in the job and optionally in the portal settings. Personal information is only stored in portal settings if you check the checkbox labeled, **Save carrier and return address as default** during the *Return shipping info* section of the export process.

Personal contact information may be deleted in the following ways:

- Data saved with the job is deleted with the job. Users can delete jobs manually and completed jobs are automatically deleted after 90 days. You can manually delete the jobs via the REST API or the Azure portal. To delete the job in the Azure portal, go to your import/export job, and click *Delete* from the command bar. For details on how to delete an import/export job via REST API, refer to [Delete an import/export job](#).
- Contact information saved in the portal settings may be removed by deleting the portal settings. You can delete portal settings by following these steps:

- Sign in to the [Azure portal](#).
- Click on the *Settings* icon 
- Click *Export all settings* (to save your current settings to a `.json` file).
- Click *Delete all settings and private dashboards* to delete all settings including saved contact information.

For more information, review the Microsoft Privacy policy at [Trust Center](#)

## Pricing

### Drive handling fee

There is a drive handling fee for each drive processed as part of your import or export job. See the details on the [Azure Import/Export Pricing](#).

### Shipping costs

When you ship drives to Azure, you pay the shipping cost to the shipping carrier. When Microsoft returns the drives to you, the shipping cost is charged to the carrier account which you provided at the time of job creation.

### Transaction costs

[Standard storage transaction charge](#) apply during import as well as export of data. Standard egress charges are also applicable along with storage transaction charges when data is exported from Azure Storage. For more information on egress costs, see [Data transfer pricing](#).

## Next steps

Learn how to use the Import/Export service to:

- [Import data to Azure Blobs](#)
- [Export data from Azure Blobs](#)
- [Import data to Azure Files](#)

# Azure Import/Export system requirements

1/14/2020 • 2 minutes to read • [Edit Online](#)

This article describes the important requirements for your Azure Import/Export service. We recommend that you review the information carefully before you use the Import/Export service and then refer back to it as necessary during the operation.

## Supported operating systems

To prepare the hard drives using the WAIImportExport tool, the following **64-bit OS that support BitLocker Drive Encryption** are supported.

PLATFORM	VERSION
Windows	Windows 7 Enterprise, Windows 7 Ultimate Windows 8 Pro, Windows 8 Enterprise, Windows 8.1 Pro, Windows 8.1 Enterprise Windows 10
Windows Server	Windows Server 2008 R2 Windows Server 2012, Windows Server 2012 R2

## Other required software for Windows client

PLATFORM	VERSION
.NET Framework	4.5.1
BitLocker	–

## Supported storage accounts

Azure Import/Export service supports the following types of storage accounts:

- Standard General Purpose v2 storage accounts (recommended for most scenarios)
- Blob Storage accounts
- General Purpose v1 storage accounts (both Classic or Azure Resource Manager deployments),

For more information about storage accounts, see [Azure storage accounts overview](#).

Each job may be used to transfer data to or from only one storage account. In other words, a single import/export job cannot span across multiple storage accounts. For information on creating a new storage account, see [How to Create a Storage Account](#).

### IMPORTANT

The Azure Import Export service does not support storage accounts where the [Virtual Network Service Endpoints](#) feature has been enabled.

## Supported storage types

The following list of storage types is supported with Azure Import/Export service.

JOB	STORAGE SERVICE	SUPPORTED	NOT SUPPORTED
Import	Azure Blob storage	Block Blobs and Page blobs supported	Azure Files not supported
	Azure File storage	Files supported	
Export	Azure Blob storage	Block blobs, Page blobs, and Append blobs supported	Azure Files not supported

## Supported hardware

For the Azure Import/Export service, you need supported disks to copy data.

### Supported disks

The following list of disks is supported for use with the Import/Export service.

DISK TYPE	SIZE	SUPPORTED
SSD	2.5"	SATA III
HDD	2.5" 3.5"	SATA II, SATA III

The following disk types are not supported:

- USBs.
- External HDD with built-in USB adaptor.
- Disks that are inside the casing of an external HDD.

A single import/export job can have:

- A maximum of 10 HDD/SSDs.
- A mix of HDD/SSD of any size.

Large number of drives can be spread across multiple jobs and there is no limits on the number of jobs that can be created. For import jobs, only the first data volume on the drive is processed. The data volume must be formatted with NTFS.

When preparing hard drives and copying the data using the WAImportExport tool, you can use external USB adaptors. Most off-the-shelf USB 3.0 or later adaptors should work.

## Next steps

- [Set up the WAImportExport tool](#)
- [Transfer data with the AzCopy command-line utility](#)
- [Azure Import Export REST API sample](#)

# Use the Azure Import/Export service to import data to Azure Blob Storage

3/25/2020 • 8 minutes to read • [Edit Online](#)

This article provides step-by-step instructions on how to use the Azure Import/Export service to securely import large amounts of data to Azure Blob storage. To import data into Azure Blobs, the service requires you to ship encrypted disk drives containing your data to an Azure datacenter.

## Prerequisites

Before you create an import job to transfer data into Azure Blob Storage, carefully review and complete the following list of prerequisites for this service. You must:

- Have an active Azure subscription that can be used for the Import/Export service.
- Have at least one Azure Storage account with a storage container. See the list of [Supported storage accounts and storage types for Import/Export service](#).
  - For information on creating a new storage account, see [How to Create a Storage Account](#).
  - For information on storage container, go to [Create a storage container](#).
- Have adequate number of disks of [Supported types](#).
- Have a Windows system running a [Supported OS version](#).
- Enable BitLocker on the Windows system. See [How to enable BitLocker](#).
- [Download the latest WAImportExport version 1](#) on the Windows system. The latest version of the tool has security updates to allow an external protector for the BitLocker key, and the updated unlock mode feature.
  - Unzip to the default folder `waimportexportv1`. For example, `c:\WaImportExportV1`.
- Have a FedEx/DHL account. If you want to use a carrier other than FedEx/DHL, contact Azure Data Box Operations team at [adbops@microsoft.com](mailto:adbops@microsoft.com).
  - The account must be valid, should have balance, and must have return shipping capabilities.
  - Generate a tracking number for the export job.
  - Every job should have a separate tracking number. Multiple jobs with the same tracking number are not supported.
  - If you do not have a carrier account, go to:
    - [Create a FedEx account](#), or
    - [Create a DHL account](#).

## Step 1: Prepare the drives

This step generates a journal file. The journal file stores basic information such as drive serial number, encryption key, and storage account details.

Perform the following steps to prepare the drives.

1. Connect your disk drives to the Windows system via SATA connectors.
2. Create a single NTFS volume on each drive. Assign a drive letter to the volume. Do not use mountpoints.

3. Enable BitLocker encryption on the NTFS volume. If using a Windows Server system, use the instructions in [How to enable BitLocker on Windows Server 2012 R2](#).
4. Copy data to encrypted volume. Use drag and drop or Robocopy or any such copy tool. A journal (*.jrn*) file is created in the same folder where you run the tool.

If the drive is locked and you need to unlock the drive, the steps to unlock may be different depending on your use case.

- If you have added data to a pre-encrypted drive (WAImportExport tool was not used for encryption), use the BitLocker key (a numerical password that you specify) in the popup to unlock the drive.
- If you have added data to a drive that was encrypted by WAImportExport tool, use the following command to unlock the drive:

```
WAImportExport Unlock /externalKey:<BitLocker key (base 64 string) copied from journal (*.jrn*) file>
```

5. Open a PowerShell or command line window with administrative privileges. To change directory to the unzipped folder, run the following command:

```
cd C:\WaImportExportV1
```

6. To get the BitLocker key of the drive, run the following command:

```
manage-bde -protectors -get <DriveLetter>:
```

7. To prepare the disk, run the following command. **Depending on the data size, this may take several hours to days.**

```
./WAImportExport.exe PrepImport /j:<journal file name> /id:<session#<session number> /t:<Drive letter> /bk:<BitLocker key> /srcdir:<Drive letter>:\ /dstdir:<Container name>/ /blobtype:<BlockBlob or PageBlob> /skipwrite
```

A Journal file is created in the same folder where you ran the tool. Two other files are also created - an *.xml* file (folder where you run the tool) and a *drive-manifest.xml* file (folder where data resides).

The parameters used are described in the following table:

OPTION	DESCRIPTION
/j:	The name of the journal file, with the <i>.jrn</i> extension. A journal file is generated per drive. We recommend that you use the disk serial number as the journal file name.
/id:	The session ID. Use a unique session number for each instance of the command.
/t:	The drive letter of the disk to be shipped. For example, drive <b>D:</b>
/bk:	The BitLocker key for the drive. Its numerical password from output of <code>manage-bde -protectors -get D:</code>
/srcdir:	The drive letter of the disk to be shipped followed by <b>:\<i>Container name</i></b> . For example, <b>D:\</b>
/dstdir:	The name of the destination container in Azure Storage.

OPTION	DESCRIPTION
/blobtype:	This option specifies the type of blobs you want to import the data to. For block blobs, this is <code>BlockBlob</code> and for page blobs, it is <code>PageBlob</code> .
/skipwrite:	The option that specifies that there is no new data required to be copied and existing data on the disk is to be prepared.
/enablecontentmd5:	The option when enabled, ensures that MD5 is computed and set as <code>Content-md5</code> property on each blob. Use this option only if you want to use the <code>Content-md5</code> field after the data is uploaded to Azure. This option does not affect the data integrity check (that occurs by default). The setting does increase the time taken to upload data to cloud.

8. Repeat the previous step for each disk that needs to be shipped. A journal file with the provided name is created for every run of the command line.

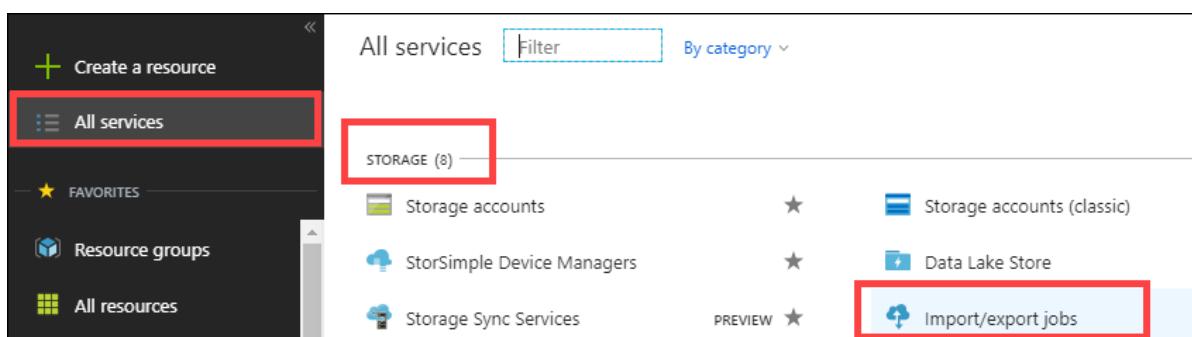
#### IMPORTANT

- Together with the journal file, a `<Journal file name>_DriveInfo_<Drive serial ID>.xml` file is also created in the same folder where the tool resides. The .xml file is used in place of journal file when creating a job if the journal file is too big.

## Step 2: Create an import job

Perform the following steps to create an import job in the Azure portal.

- Log on to <https://portal.azure.com/>.
- Go to All services > Storage > Import/export jobs.



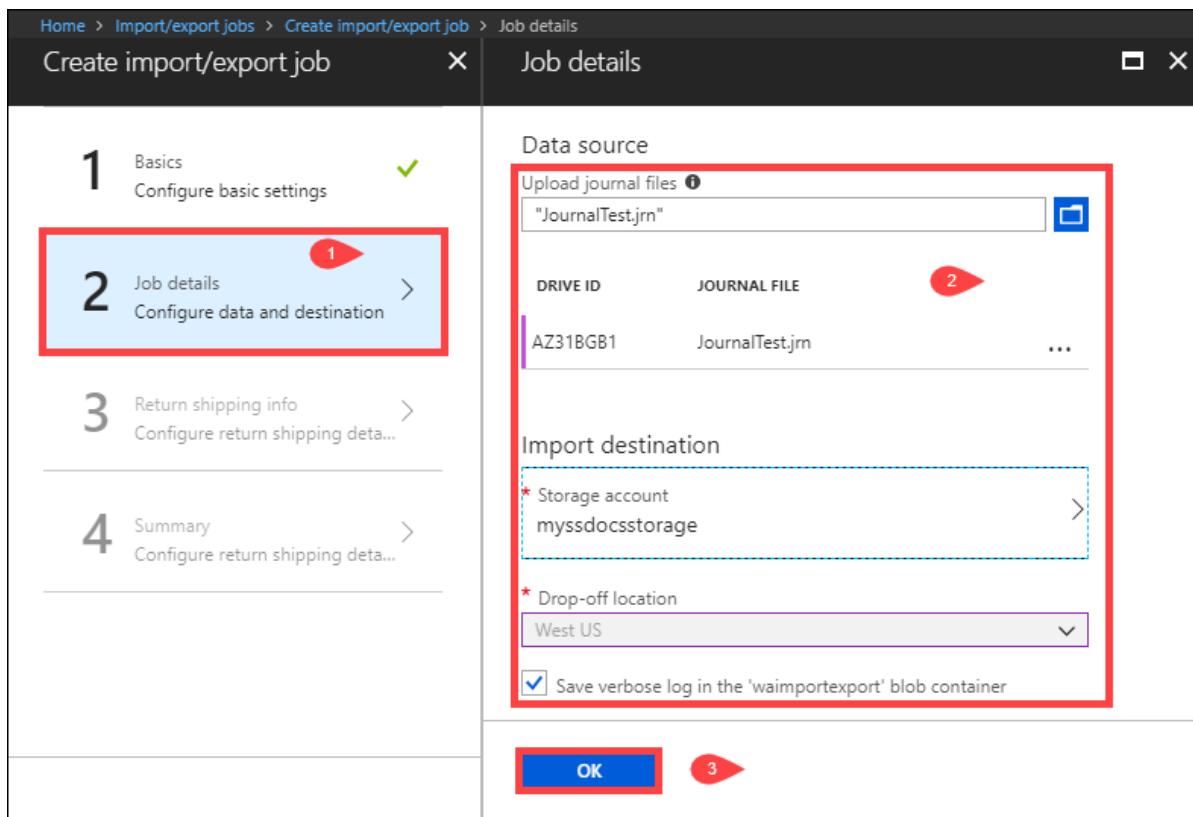
- Click **Create Import/export Job**.

#### 4. In Basics:

- Select **Import into Azure**.
- Enter a descriptive name for the import job. Use the name to track the progress of your jobs.
  - The name may contain only lowercase letters, numbers, and hyphens.
  - The name must start with a letter, and may not contain spaces.
- Select a subscription.
- Enter or select a resource group.

#### 5. In Job details:

- Upload the drive journal files that you obtained during the drive preparation step. If `waimportexport.exe version1` was used, upload one file for each drive that you prepared. If the journal file size exceeds 2 MB, then you can use the `<Journal file name>_DriveInfo_<Drive serial ID>.xml` also created with the journal file.
- Select the destination storage account where data will reside.
- The dropoff location is automatically populated based on the region of the storage account selected.



#### 6. In Return shipping info:

- Select the carrier from the dropdown list. If you want to use a carrier other than FedEx/DHL, choose an existing option from the dropdown. Contact Azure Data Box Operations team at [adbops@microsoft.com](mailto:adbops@microsoft.com) with the information regarding the carrier you plan to use.
- Enter a valid carrier account number that you have created with that carrier. Microsoft uses this account to ship the drives back to you once your import job is complete. If you do not have an account number, create a [FedEx](#) or [DHL](#) carrier account.
- Provide a complete and valid contact name, phone, email, street address, city, zip, state/province and country/region.

#### TIP

Instead of specifying an email address for a single user, provide a group email. This ensures that you receive notifications even if an admin leaves.

**Create import/export job**

- 1 Basics** ✓  
Configure basic settings
- 2 Job details** ✓  
Configure data and destination
- 3 Return shipping info** ✓  
Configure return shipping data... 1
- 4 Summary** ✓  
Configure return shipping data... 2

**Return shipping info**

**Return carrier**

\* Carrier name: FedEx

\* Carrier account number: 123456789

**Return address**

\* Contact name: Gus Poland

\* Phone: 4085555555

\* Email: gus@contoso.com

\* Street address 1: 1020 Enterprise Way

Street address 2 (optional): 3

\* City: Sunnyvale

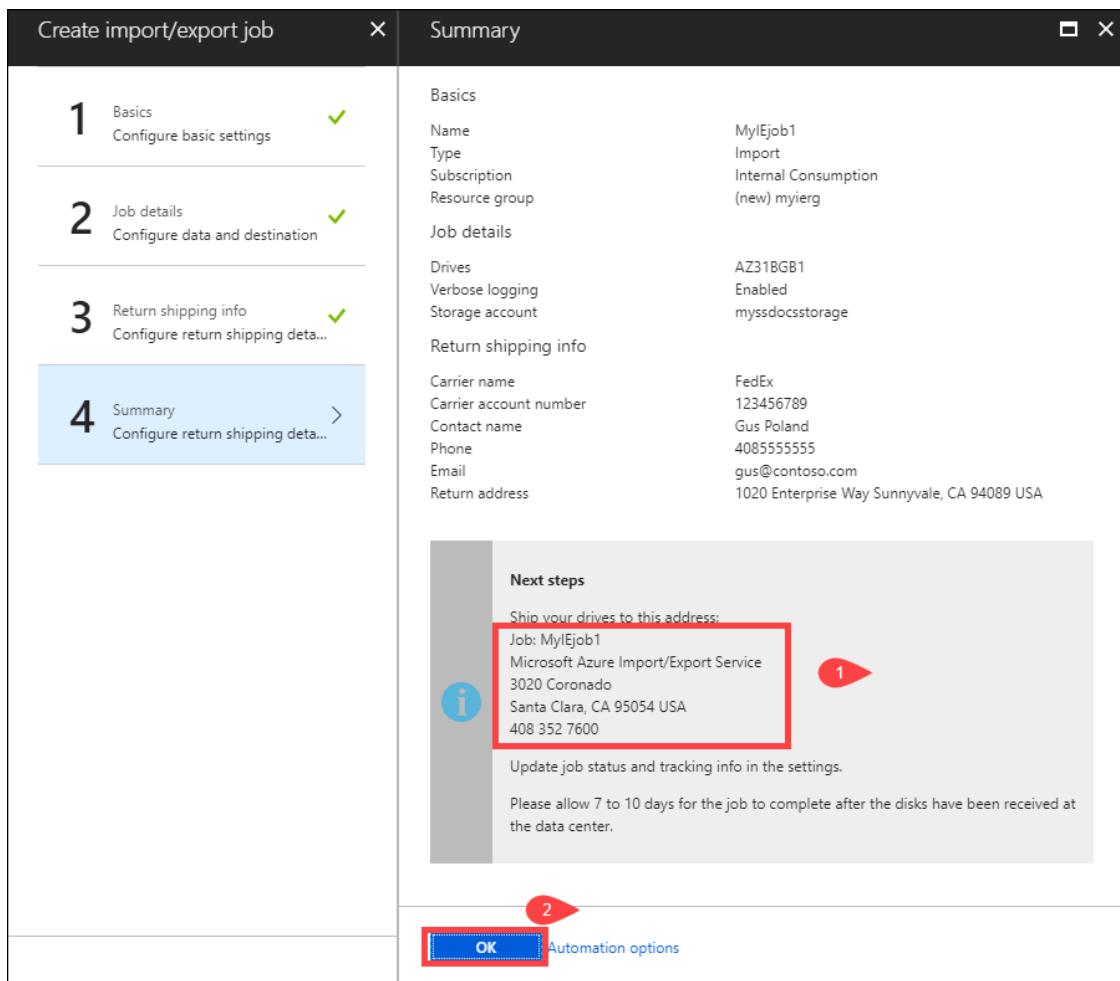
* Zip code: 94089	State/Province: CA
* Country/Region: USA	

Save carrier and return address as default.

**OK**

7. In the **Summary**:

- Review the job information provided in the summary. Make a note of the job name and the Azure datacenter shipping address to ship disks back to Azure. This information is used later on the shipping label.
- Click **OK** to create the import job.



## Step 3 (Optional): Configure customer managed key

Skip this step and go to the next step if you want to use the Microsoft managed key to protect your BitLocker keys for the drives. To configure your own key to protect the BitLocker key, follow the instructions in [Configure customer-managed keys with Azure Vault for Azure Import/Export in the Azure portal](#)

## Step 4: Ship the drives

FedEx, UPS, or DHL can be used to ship the package to Azure datacenter. If you want to use a carrier other than FedEx/DHL, contact Azure Data Box Operations team at [adbops@microsoft.com](mailto:adbops@microsoft.com)

- Provide a valid FedEx, UPS, or DHL carrier account number that Microsoft will use to ship the drives back.
  - A FedEx, UPS, or DHL account number is required for shipping drives back from the US and Europe locations.
  - A DHL account number is preferred for shipping drives back from Asia and Australia locations.
  - If you do not have an account number, create a [FedEx](#) or [DHL](#) carrier account.
- When shipping your packages, you must follow the [Microsoft Azure Service Terms](#).
- Properly package yours disks to avoid potential damage and delays in processing.

## Step 5: Update the job with tracking information

After shipping the disks, return to the **Import/Export** page on the Azure portal.

### IMPORTANT

If the tracking number is not updated within 2 weeks of creating the job, the job expires.

To update the tracking number, perform the following steps.

1. Select and click the job.
2. Click **Update job status and tracking info once drives are shipped**.
3. Select the checkbox against **Mark as shipped**.
4. Provide the **Carrier and Tracking number**.
5. Track the job progress on the portal dashboard. For a description of each job state, go to [View your job status](#).

## Step 6: Verify data upload to Azure

Track the job to completion. Once the job is complete, verify that your data has uploaded to Azure. Delete the on-premises data only after you have verified that upload was successful.

### Next steps

- [View the job and drive status](#)
- [Review Import/Export requirements](#)

# Use Azure Import/Export service to import data to Azure Files

1/15/2020 • 7 minutes to read • [Edit Online](#)

This article provides step-by-step instructions on how to use the Azure Import/Export service to securely import large amounts of data into Azure Files. To import data, the service requires you to ship supported disk drives containing your data to an Azure datacenter.

The Import/Export service supports only import of Azure Files into Azure Storage. Exporting Azure Files is not supported.

## Prerequisites

Before you create an import job to transfer data into Azure Files, carefully review and complete the following list of prerequisites. You must:

- Have an active Azure subscription to use with Import/Export service.
- Have at least one Azure Storage account. See the list of [Supported storage accounts and storage types for Import/Export service](#). For information on creating a new storage account, see [How to Create a Storage Account](#).
- Have adequate number of disks of [Supported types](#).
- Have a Windows system running a [Supported OS version](#).
- [Download the WAImportExport version 2](#) on the Windows system. Unzip to the default folder `waimportexport`.  
For example, `C:\WaImportExport`.
- Have a FedEx/DHL account. If you want to use a carrier other than FedEx/DHL, contact Azure Data Box Operations team at `adbops@microsoft.com`.
  - The account must be valid, should have balance, and must have return shipping capabilities.
  - Generate a tracking number for the export job.
  - Every job should have a separate tracking number. Multiple jobs with the same tracking number are not supported.
  - If you do not have a carrier account, go to:
    - [Create a FedEx account](#), or
    - [Create a DHL account](#).

## Step 1: Prepare the drives

This step generates a journal file. The journal file stores basic information such as drive serial number, encryption key, and storage account details.

Perform the following steps to prepare the drives.

1. Connect our disk drives to the Windows system via SATA connectors.
2. Create a single NTFS volume on each drive. Assign a drive letter to the volume. Do not use mountpoints.
3. Modify the `dataset.csv` file in the root folder where the tool resides. Depending on whether you want to import a file or folder or both, add entries in the `dataset.csv` file similar to the following examples.
  - **To import a file:** In the following example, the data to copy resides in the F: drive. Your file `MyFile1.txt` is copied to the root of the `MyAzureFileshare1`. If the `MyAzureFileshare1` does not exist, it

is created in the Azure Storage account. Folder structure is maintained.

```
BasePath,DstItemPathOrPrefix,ItemType,Disposition,MetadataFile,PropertiesFile
"F:\MyFolder1\MyFile1.txt","MyAzureFileshare1/MyFile1.txt",file,rename,"None",None
```

- **To import a folder:** All files and folders under *MyFolder2* are recursively copied to fileshare. Folder structure is maintained.

```
"F:\MyFolder2\" , "MyAzureFileshare1/" , file , rename , "None" , None
```

Multiple entries can be made in the same file corresponding to folders or files that are imported.

```
"F:\MyFolder1\MyFile1.txt","MyAzureFileshare1/MyFile1.txt",file,rename,"None",None
"F:\MyFolder2\" , "MyAzureFileshare1/" , file , rename , "None" , None
```

Learn more about [preparing the dataset CSV file](#).

4. Modify the *driveset.csv* file in the root folder where the tool resides. Add entries in the *driveset.csv* file similar to the following examples. The driveset file has the list of disks and corresponding drive letters so that the tool can correctly pick the list of disks to be prepared.

This example assumes that two disks are attached and basic NTFS volumes G:\ and H:\ are created. H:\ is not encrypted while G: is already encrypted. The tool formats and encrypts the disk that hosts H:\ only (and not G:).

- **For a disk that is not encrypted:** Specify *Encrypt* to enable BitLocker encryption on the disk.

```
DriveLetter,FormatOption,SilentOrPromptOnFormat,Encryption,ExistingBitLockerKey
H,Format,SilentMode,Encrypt,
```

- **For a disk that is already encrypted:** Specify *AlreadyEncrypted* and supply the BitLocker key.

```
DriveLetter,FormatOption,SilentOrPromptOnFormat,Encryption,ExistingBitLockerKey
G,AlreadyFormatted,SilentMode,AlreadyEncrypted,060456-014509-132033-080300-252615-584177-672089-
411631
```

Multiple entries can be made in the same file corresponding to multiple drives. Learn more about [preparing the driveset CSV file](#).

5. Use the `PrepImport` option to copy and prepare data to the disk drive. For the first copy session to copy directories and/or files with a new copy session, run the following command:

```
...
.\\WAIImportExport.exe PrepImport /j:<JournalFile> /id:<SessionId> [/logdir:<LogDirectory>] [/sk:<StorageAccountKey>] [/silentmode] [/InitialDriveSet:<driveset.csv>] DataSet:<dataset.csv>
...
```

An import example is shown below.

```
...
.\WAIImportExport.exe PrepImport /j:JournalTest.jrn /id:session#1 /sk:*****
/InitialDriveSet:driveset.csv /DataSet:dataset.csv /logdir:C:\logs
...
```

6. A journal file with name you provided with `/j:` parameter, is created for every run of the command line. Each drive you prepare has a journal file that must be uploaded when you create the import job. Drives without journal files are not processed.

**IMPORTANT**

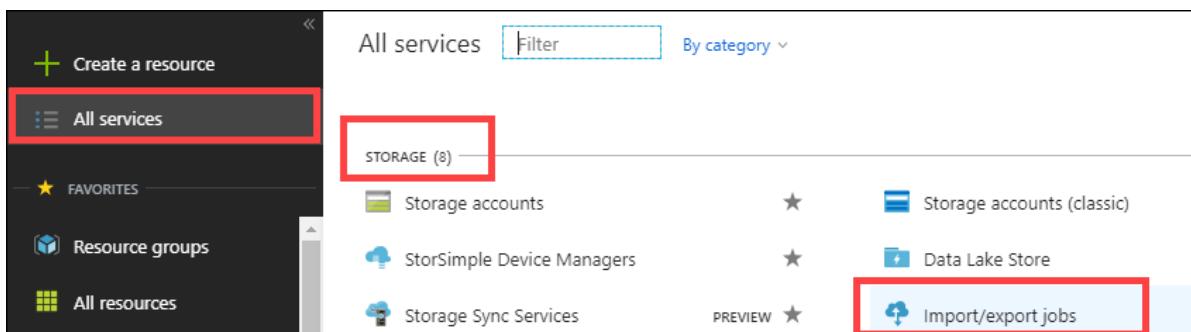
- Do not modify the data on the disk drives or the journal file after completing disk preparation.

For additional samples, go to [Samples for journal files](#).

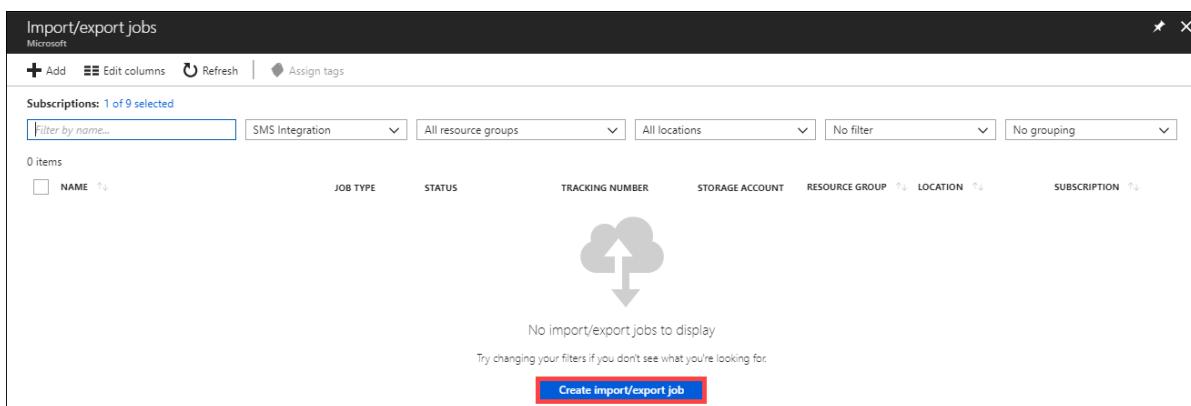
## Step 2: Create an import job

Perform the following steps to create an import job in the Azure portal.

1. Log on to <https://portal.azure.com/>.
2. Go to All services > Storage > Import/export jobs.



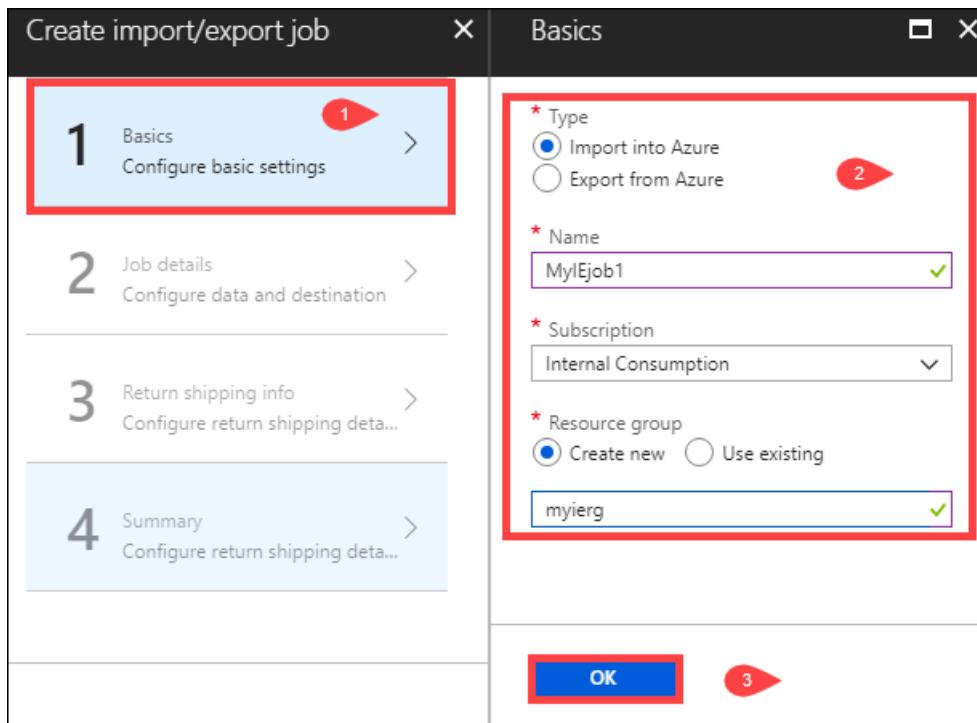
3. Click **Create Import/export Job**.



4. In Basics:

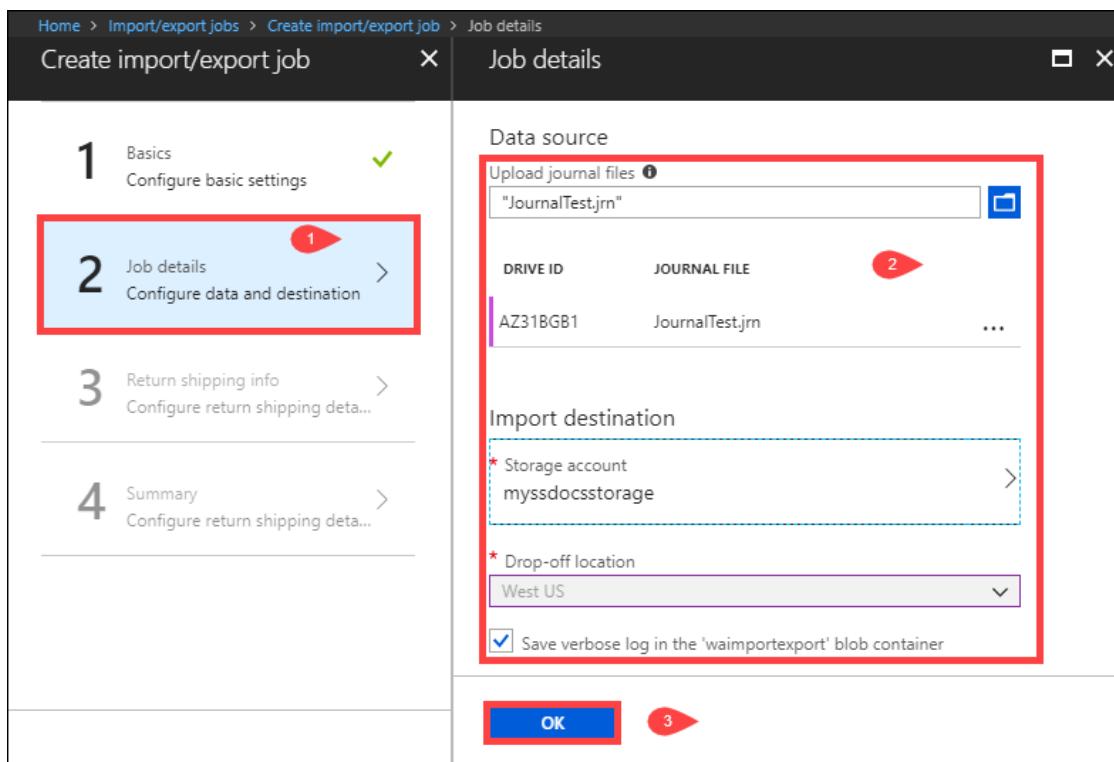
- Select **Import into Azure**.
- Enter a descriptive name for the import job. Use this name to track your jobs while they are in progress and once they are completed.
  - This name may contain only lowercase letters, numbers, hyphens, and underscores.
  - The name must start with a letter, and may not contain spaces.
- Select a subscription.

- Select a resource group.



#### 5. In Job details:

- Upload the journal files that you created during the preceding [Step 1: Prepare the drives](#).
- Select the storage account that the data will be imported into.
- The dropoff location is automatically populated based on the region of the storage account selected.



#### 6. In Return shipping info:

- Select the carrier from the drop-down list. If you want to use a carrier other than FedEx/DHL, choose an existing option from the dropdown. Contact Azure Data Box Operations team at [adbops@microsoft.com](mailto:adbops@microsoft.com) with the information regarding the carrier you plan to use.

- Enter a valid carrier account number that you have created with that carrier. Microsoft uses this account to ship the drives back to you once your import job is complete.
- Provide a complete and valid contact name, phone, email, street address, city, zip, state/province and country/region.

**TIP**

Instead of specifying an email address for a single user, provide a group email. This ensures that you receive notifications even if an admin leaves.

The screenshot shows the 'Create import/export job' wizard with four steps:

- 1 Basics**: Configure basic settings (Completed)
- 2 Job details**: Configure data and destination (Completed)
- 3 Return shipping info**: Configure return shipping data... (Selected, highlighted with a red box)
- 4 Summary**: Configure return shipping data... (Next step, indicated by a red arrow)

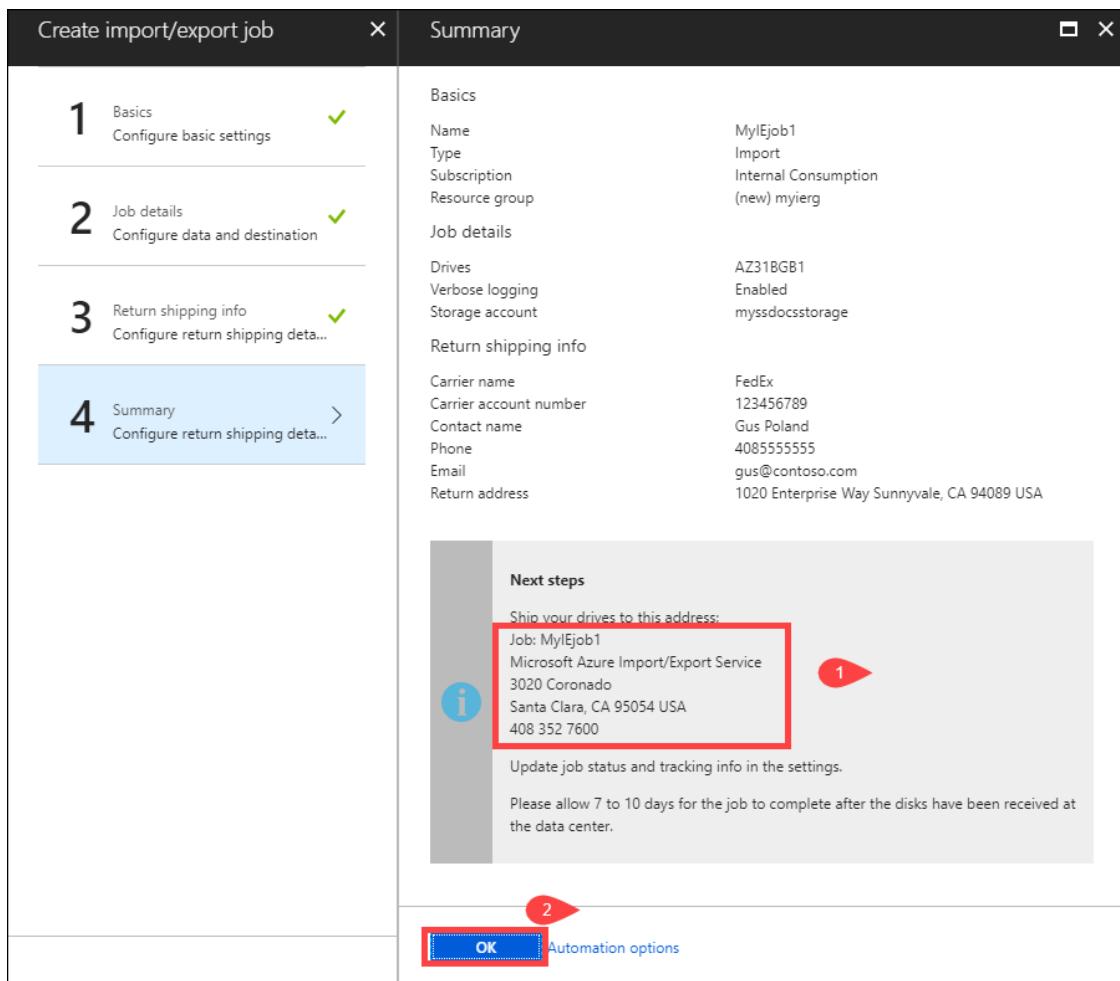
In the 'Return shipping info' step, the 'Return carrier' section is expanded. It includes:

- Carrier name:** FedEx
- Carrier account number:** 123456789
- Contact name:** Gus Poland
- Phone:** 4085555555
- Email:** gus@contoso.com
- Street address 1:** 1020 Enterprise Way (highlighted with a red box and arrow 1)
- Street address 2 (optional):** (highlighted with a red box and arrow 2)
- City:** Sunnyvale
- Zip code:** 94089
- State/Province:** CA
- Country/Region:** USA
- Save carrier and return address as default:** (checkbox)

A large red box surrounds the 'Carrier name' dropdown and its associated fields. Red arrows point from the 'Street address 1' field (arrow 1) and the 'OK' button (arrow 3) towards the right margin.

7. In the **Summary**:

- Provide the Azure datacenter shipping address for shipping disks back to Azure. Ensure that the job name and the full address are mentioned on the shipping label.
- Click **OK** to complete import job creation.



## Step 3: Ship the drives to the Azure datacenter

FedEx, UPS, or DHL can be used to ship the package to Azure datacenter. If you want to use a carrier other than FedEx/DHL, contact Azure Data Box Operations team at [adbops@microsoft.com](mailto:adbops@microsoft.com)

- Provide a valid FedEx, UPS, or DHL carrier account number that Microsoft will use to ship the drives back.
  - A FedEx, UPS, or DHL account number is required for shipping drives back from the US and Europe locations.
  - A DHL account number is preferred for shipping drives back from Asia and Australia locations.
  - If you do not have an account number, create a [FedEx](#) or [DHL](#) carrier account.
- When shipping your packages, you must follow the [Microsoft Azure Service Terms](#).
- Properly package yours disks to avoid potential damage and delays in processing.

## Step 4: Update the job with tracking information

After shipping the disks, return to the **Import/Export** page on the Azure portal.

### IMPORTANT

If the tracking number is not updated within 2 weeks of creating the job, the job expires.

To update the tracking number, perform the following steps.

1. Select and click the job.
2. Click **Update job status and tracking info once drives are shipped**.
3. Select the checkbox against **Mark as shipped**.

4. Provide the Carrier and Tracking number.
5. Track the job progress on the portal dashboard. For a description of each job state, go to [View your job status](#).

## Step 5: Verify data upload to Azure

Track the job to completion. Once the job is complete, verify that your data has uploaded to Azure. Delete the on-premises data only after you have verified that upload was successful.

### Samples for journal files

To **add more drives**, create a new driveset file and run the command as below.

For subsequent copy sessions to the different disk drives than specified in *InitialDriveset.csv* file, specify a new driveset .csv file and provide it as a value to the parameter `AdditionalDriveSet`. Use the **same journal file name** and provide a **new session ID**. The format of AdditionalDriveset CSV file is same as InitialDriveSet format.

```
...
WAIImportExport.exe PrepImport /j:<JournalFile> /id:<SessionId> /AdditionalDriveSet:<driveset.csv>
...
```

An import example is shown below.

```
...
WAIImportExport.exe PrepImport /j:JournalTest.jrn /id:session#3 /AdditionalDriveSet:driveset-2.csv
...
```

To add additional data to the same driveset, use the PreImport command for subsequent copy sessions to copy additional files/directory.

For subsequent copy sessions to the same hard disk drives specified in *InitialDriveset.csv* file, specify the **same journal file name** and provide a **new session ID**; there is no need to provide the storage account key.

```
...
WAIImportExport PrepImport /j:<JournalFile> /id:<SessionId> /j:<JournalFile> /id:<SessionId> [/logdir:<LogDirectory>] DataSet:<dataset.csv>
...
```

An import example is shown below.

```
...
WAIImportExport.exe PrepImport /j:JournalTest.jrn /id:session#2 /DataSet:dataset-2.csv
...
```

## Next steps

- [View the job and drive status](#)
- [Review Import/Export requirements](#)

# Use the Azure Import/Export service to export data from Azure Blob storage

3/25/2020 • 8 minutes to read • [Edit Online](#)

This article provides step-by-step instructions on how to use the Azure Import/Export service to securely export large amounts of data from Azure Blob storage. The service requires you to ship empty drives to the Azure datacenter. The service exports data from your storage account to the drives and then ships the drives back.

## Prerequisites

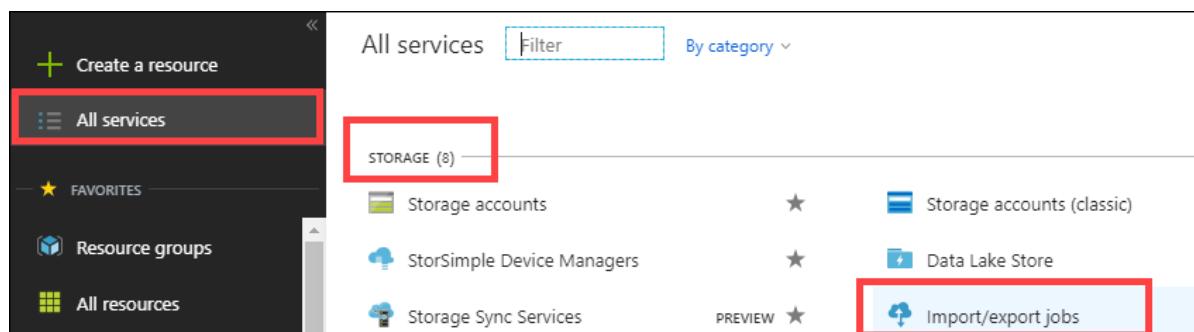
Before you create an export job to transfer data out of Azure Blob Storage, carefully review and complete the following list of prerequisites for this service. You must:

- Have an active Azure subscription that can be used for the Import/Export service.
- Have at least one Azure Storage account. See the list of [Supported storage accounts and storage types for Import/Export service](#). For information on creating a new storage account, see [How to Create a Storage Account](#).
- Have adequate number of disks of [Supported types](#).
- Have a FedEx/DHL account. If you want to use a carrier other than FedEx/DHL, contact Azure Data Box Operations team at [adbops@microsoft.com](mailto:adbops@microsoft.com).
  - The account must be valid, should have balance, and must have return shipping capabilities.
  - Generate a tracking number for the export job.
  - Every job should have a separate tracking number. Multiple jobs with the same tracking number are not supported.
  - If you do not have a carrier account, go to:
    - [Create a FedEx account](#), or
    - [Create a DHL account](#).

## Step 1: Create an export job

Perform the following steps to create an export job in the Azure portal.

1. Log on to <https://portal.azure.com/>.
2. Go to All services > Storage > Import/export jobs.



3. Click **Create Import/export Job**.

The screenshot shows the 'Import/export jobs' page in the Azure portal. At the top, there are buttons for 'Add', 'Edit columns', 'Refresh', and 'Assign tags'. Below that, a message says 'Subscriptions: 1 of 9 selected'. There are filter options for 'Filter by name...', 'SMS Integration', 'All resource groups', 'All locations', 'No filter', and 'No grouping'. A large central area displays a cloud icon with an upward arrow and a downward arrow, indicating no import/export jobs are currently displayed. Below the icon, it says 'No import/export jobs to display' and 'Try changing your filters if you don't see what you're looking for'. A red button at the bottom right says 'Create import/export job'.

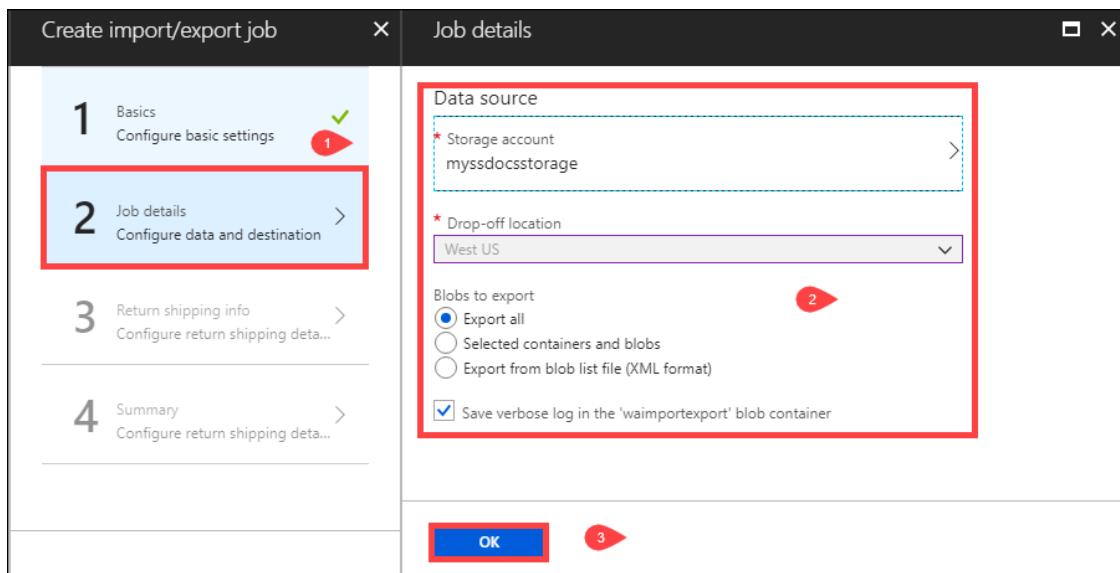
#### 4. In Basics:

- Select **Export from Azure**.
- Enter a descriptive name for the export job. Use the name you choose to track the progress of your jobs.
  - The name may contain only lowercase letters, numbers, hyphens, and underscores.
  - The name must start with a letter, and may not contain spaces.
- Select a subscription.
- Enter or select a resource group.

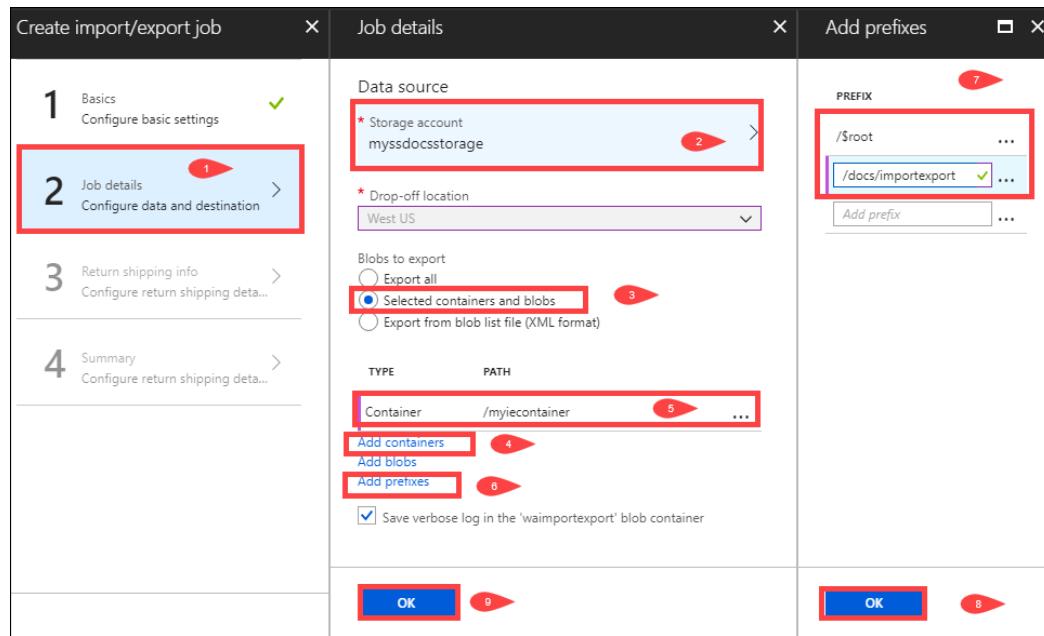
The screenshot shows the 'Create import/export job' wizard. On the left, a vertical navigation pane lists four steps: 1. Basics (selected), 2. Job details, 3. Return shipping info, and 4. Summary. Step 1 is highlighted with a red box and a red numbered callout '1'. On the right, the 'Basics' configuration pane is shown. It includes fields for 'Type' (set to 'Export from Azure'), 'Name' ('Myexportjob1'), 'Subscription' ('Internal Consumption'), and 'Resource group' ('myierg'). The 'Resource group' field is also highlighted with a red box and a red numbered callout '2'. At the bottom right of the pane is a blue 'OK' button, which is also highlighted with a red box and a red numbered callout '3'.

#### 5. In Job details:

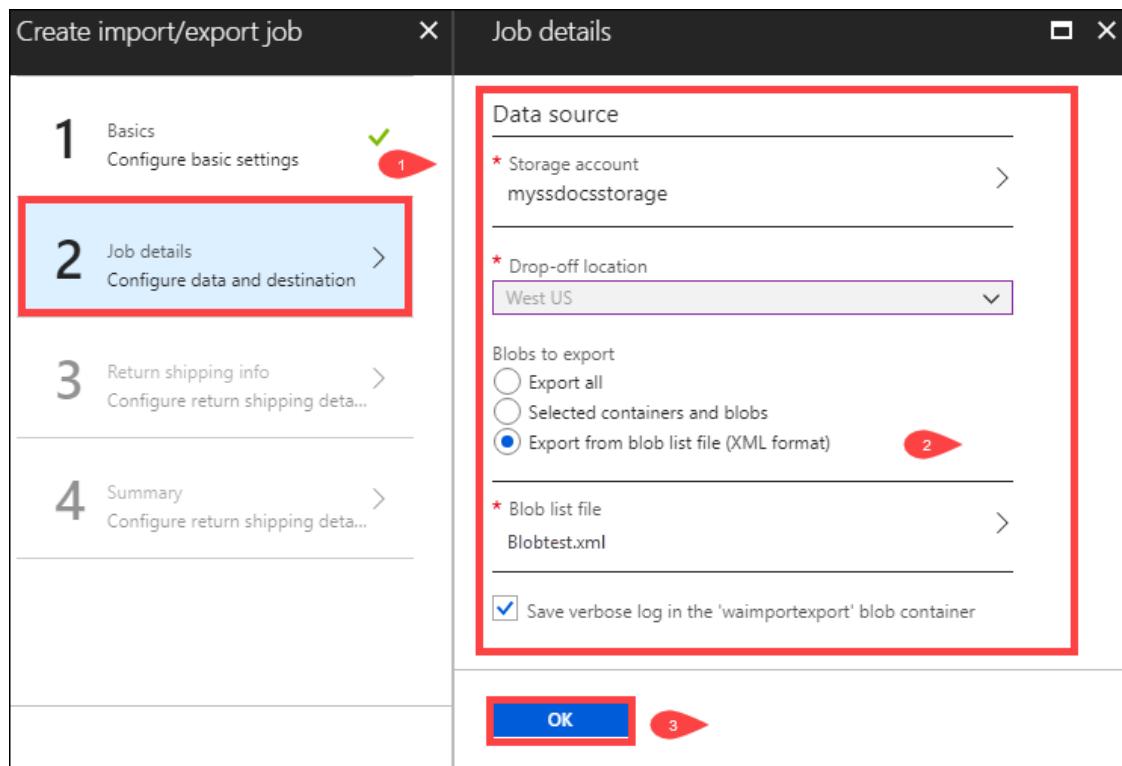
- Select the storage account where the data to be exported resides. Use a storage account close to where you are located.
- The dropoff location is automatically populated based on the region of the storage account selected.
- Specify the blob data you wish to export from your storage account to your blank drive or drives.
- Choose to **Export all** blob data in the storage account.



- You can specify which containers and blobs to export.
  - **To specify a blob to export:** Use the **Equal To** selector. Specify the relative path to the blob, beginning with the container name. Use **\$root** to specify the root container.
  - **To specify all blobs starting with a prefix:** Use the **Starts With** selector. Specify the prefix, beginning with a forward slash '/'. The prefix may be the prefix of the container name, the complete container name, or the complete container name followed by the prefix of the blob name. You must provide the blob paths in valid format to avoid errors during processing, as shown in this screenshot. For more information, see [Examples of valid blob paths](#).



- You can export from the blob list file.



#### NOTE

If the blob to be exported is in use during data copy, Azure Import/Export service takes a snapshot of the blob and copies the snapshot.

#### 6. In Return shipping info:

- Select the carrier from the dropdown list. If you want to use a carrier other than FedEx/DHL, choose an existing option from the dropdown. Contact Azure Data Box Operations team at [adbops@microsoft.com](mailto:adbops@microsoft.com) with the information regarding the carrier you plan to use.
- Enter a valid carrier account number that you have created with that carrier. Microsoft uses this account to ship the drives back to you once your export job is complete.
- Provide a complete and valid contact name, phone, email, street address, city, zip, state/province and country/region.

#### TIP

Instead of specifying an email address for a single user, provide a group email. This ensures that you receive notifications even if an admin leaves.

#### 7. In Summary:

- Review the details of the job.
- Make a note of the job name and provided Azure datacenter shipping address for shipping disks to Azure.

#### NOTE

Always send the disks to the datacenter noted in the Azure portal. If the disks are shipped to the wrong datacenter, the job will not be processed.

- Click **OK** to complete export job creation.

## Step 2: Ship the drives

If you do not know the number of drives you need, go to the [Check the number of drives](#). If you know the number of drives, proceed to ship the drives.

FedEx, UPS, or DHL can be used to ship the package to Azure datacenter. If you want to use a carrier other than FedEx/DHL, contact Azure Data Box Operations team at [adbops@microsoft.com](mailto:adbops@microsoft.com)

- Provide a valid FedEx, UPS, or DHL carrier account number that Microsoft will use to ship the drives back.
  - A FedEx, UPS, or DHL account number is required for shipping drives back from the US and Europe locations.
  - A DHL account number is preferred for shipping drives back from Asia and Australia locations.
  - If you do not have an account number, create a [FedEx](#) or [DHL](#) carrier account.
- When shipping your packages, you must follow the [Microsoft Azure Service Terms](#).
- Properly package yours disks to avoid potential damage and delays in processing.

## Step 3: Update the job with tracking information

After shipping the disks, return to the **Import/Export** page on the Azure portal.

### IMPORTANT

If the tracking number is not updated within 2 weeks of creating the job, the job expires.

To update the tracking number, perform the following steps.

1. Select and click the job.
2. Click **Update job status and tracking info once drives are shipped**.
3. Select the checkbox against **Mark as shipped**.
4. Provide the **Carrier** and **Tracking number**.
5. Track the job progress on the portal dashboard. For a description of each job state, go to [View your job status](#).

## Step 4: Receive the disks

When the dashboard reports the job is complete, the disks are shipped to you and the tracking number for the shipment is available on the portal.

1. After you receive the drives with exported data, you need to get the BitLocker keys to unlock the drives. Go to the export job in the Azure portal. Click **Import/Export** tab.
2. Select and click your export job from the list. Go to **Encryption** and copy the keys.

Device BitLocker key

Drive ID	BitLocker key
AB0123456789	123456-123456-123456-123456-123456-123456-123456...

Encryption type

Your BitLocker key is protected by default using a Microsoft managed key. Select the Customer managed key to use your own key from the key vault.

[Learn more about customer managed keys.](#)

Select type

Microsoft managed key  
 Customer managed key

Current key

<https://<usercomputername>.vault.azure.net/keys/key4/e0000000ed448329adf305dd7be9655>

3. Use the BitLocker keys to unlock the disks.

The export is complete.

## Step 5: Unlock the disks

If using version 1.4.0.300 of the WAImportExport tool, use the following command to unlock the drive:

```
`WAImportExport Unlock /externalKey:<BitLocker key (base 64 string) copied from journal (*.jrn*) file>`
```

If using earlier versions of the tool, use the BitLocker dialog to unlock the drive.

At this time, you can delete the job or leave it. Jobs automatically get deleted after 90 days.

## Check the number of drives

This *optional* step helps you determine the number of drives required for the export job. Perform this step on a Windows system running a [Supported OS version](#).

1. [Download the WAImportExport version 1](#) on the Windows system.
2. Unzip to the default folder `waimportexportv1`. For example, `C:\WaImportExportV1`.
3. Open a PowerShell or command line window with administrative privileges. To change directory to the unzipped folder, run the following command:

```
cd C:\WaImportExportV1
```

- To check the number of disks required for the selected blobs, run the following command:

```
WAImportExport.exe PreviewExport /sn:<Storage account name> /sk:<Storage account key>
/ExportBlobListFile:<Path to XML blob list file> /DriveSize:<Size of drives used>
```

The parameters are described in the following table:

COMMAND-LINE PARAMETER	DESCRIPTION
/logdir:	Optional. The log directory. Verbose log files are written to this directory. If not specified, the current directory is used as the log directory.
/sn:	Required. The name of the storage account for the export job.
/sk:	Required only if a container SAS is not specified. The account key for the storage account for the export job.
/csas:	Required only if a storage account key is not specified. The container SAS for listing the blobs to be exported in the export job.
/ExportBlobListFile:	Required. Path to the XML file containing list of blob paths or blob path prefixes for the blobs to be exported. The file format used in the <code>BlobListBlobPath</code> element in the <a href="#">Put Job</a> operation of the Import/Export service REST API.
/DriveSize:	Required. The size of drives to use for an export job, <i>e.g.</i> , 500 GB, 1.5 TB.

See an [Example of the PreviewExport command](#).

- Check that you can read/write to the drives that will be shipped for the export job.

### Example of PreviewExport command

The following example demonstrates the `PreviewExport` command:

```
WAImportExport.exe PreviewExport /sn:bobmediaaccount
/sk:VkGbrUqBWLJ6zg1m29VOTrxpBgdN0lp+kp0C9MEdx3GELxmBw4hK94f7KysbbeKLDksg7VoN1W/a5UuM2zNgQ==
/ExportBlobListFile:C:\WAImportExport\mybloblist.xml /DriveSize:500GB
```

The export blob list file may contain blob names and blob prefixes, as shown here:

```
<?xml version="1.0" encoding="utf-8"?>
<BlobList>
<BlobPath>pictures/animals/koala.jpg</BlobPath>
<BlobPathPrefix>/vhds/</BlobPathPrefix>
<BlobPathPrefix>/movies/</BlobPathPrefix>
</BlobList>
```

The Azure Import/Export Tool lists all blobs to be exported and calculates how to pack them into drives of the specified size, taking into account any necessary overhead, then estimates the number of drives needed to hold the blobs and drive usage information.

Here is an example of the output, with informational logs omitted:

```

Number of unique blob paths/prefixes: 3
Number of duplicate blob paths/prefixes: 0
Number of nonexistent blob paths/prefixes: 1

Drive size: 500.00 GB
Number of blobs that can be exported: 6
Number of blobs that cannot be exported: 2
Number of drives needed: 3
 Drive #1: blobs = 1, occupied space = 454.74 GB
 Drive #2: blobs = 3, occupied space = 441.37 GB
 Drive #3: blobs = 2, occupied space = 131.28 GB

```

## Examples of valid blob paths

The following table shows examples of valid blob paths:

SELECTOR	BLOB PATH	DESCRIPTION
Starts With	/	Exports all blobs in the storage account
Starts With	/\$root/	Exports all blobs in the root container
Starts With	/book	Exports all blobs in any container that begins with prefix <b>book</b>
Starts With	/music/	Exports all blobs in container <b>music</b>
Starts With	/music/love	Exports all blobs in container <b>music</b> that begin with prefix <b>love</b>
Equal To	\$root/logo.bmp	Exports blob <b>logo.bmp</b> in the root container
Equal To	videos/story.mp4	Exports blob <b>story.mp4</b> in container <b>videos</b>

## Next steps

- [View the job and drive status](#)
- [Review Import/Export requirements](#)

# Use customer-managed keys in Azure Key Vault for Import/Export service

4/16/2020 • 4 minutes to read • [Edit Online](#)

Azure Import/Export protects the BitLocker keys used to lock the drives via an encryption key. By default, BitLocker keys are encrypted with Microsoft-managed keys. For additional control over encryption keys, you can also provide customer-managed keys.

Customer-managed keys must be created and stored in an Azure Key Vault. For more information about Azure Key Vault, see [What is Azure Key Vault?](#)

This article shows how to use customer-managed keys with Import/Export service in the [Azure portal](#).

## Prerequisites

Before you begin, make sure:

1. You have created an import or an export job as per the instructions in:
  - [Create an import job for blobs](#).
  - [Create an import job for files](#).
  - [Create an export job for blobs](#)
2. You have an existing Azure Key Vault with a key in it that you can use to protect your BitLocker key. To learn how to create a key vault using the Azure portal, see [Quickstart: Set and retrieve a secret from Azure Key Vault using the Azure portal](#).
  - **Soft delete** and **Do not purge** are set on your existing Key Vault. These properties are not enabled by default. To enable these properties, see the sections titled **Enabling soft-delete** and **Enabling Purge Protection** in one of the following articles:
    - [How to use soft-delete with PowerShell](#).
    - [How to use soft-delete with CLI](#).
  - The existing key vault should have an RSA key of 2048 size or more. For more information about keys, see **Key Vault keys** in [About Azure Key Vault keys, secrets, and certificates](#).
  - Key vault must be in the same region as the storage account for your data.
  - If you don't have an existing Azure Key Vault, you can also create it inline as described in the following section.

## Enable keys

Configuring customer-managed key for your Import/Export service is optional. By default, the Import/Export service uses a Microsoft managed key to protect your BitLocker key. To enable customer-managed keys in the Azure portal, follow these steps:

1. Go to the **Overview** blade for your Import job.
2. In the right-pane, select **Choose how your BitLocker keys are encrypted**.

**BugBashImport1**  
Import/export job

Overview

Activity log  
Access control (IAM)  
Tags  
Diagnose and solve problems

Settings

- Manage shipping info
- Encryption**
- Properties
- Locks
- Export template

Monitoring

- Diagnostics

Support + troubleshooting

New support request

Resource group (change) CMK-BUGBASH-0503

Job status Creating

Location West India

Subscription (change) <Subscription name>

Subscription ID <Subscription ID>

Overall percent completed -

Storage account xtwebscoutsa

Type Import

Delivery tracking number -

Return tracking number -

**1 drive**

Drive ID	Drive ID	Copy status	Percent complete
WD-AFAKEID1	Specified	-	-

3. In the **Encryption** blade, you can view and copy the device BitLocker key. Under **Encryption type**, you can choose how you want to protect your BitLocker key. By default, a Microsoft managed key is used.

**BugBashImport1 | Encryption**  
Import/export job

Search (Ctrl+ /)

Save ...

Device BitLocker key

Drive ID	BitLocker key
WD-AFAKEID1	<BitLocker key>

Encryption type

Your BitLocker key is protected by default using a Microsoft managed key. Select the Customer managed key to use your own key from the key vault.  
[Learn more about customer managed keys.](#)

After you select the Customer managed key, you can't switch to Microsoft managed key. If Customer managed key is chosen, your job 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.

Select type

Microsoft managed key  
 Customer managed key

4. You have the option to specify a customer managed key. After you have selected the customer managed key, **Select key vault and a key**.

Home > BugBashImport1 | Encryption

## BugBashImport1 | Encryption

Import/export job

Search (Ctrl+ /) <>

Save ...

Device BitLocker key

Drive ID	BitLocker key
WD-AFAKEID1	<BitLocker key>

Encryption type

Your BitLocker key is protected by default using a Microsoft managed key. Select the Customer managed key to use your own key from the key vault.  
[Learn more about customer managed keys.](#)

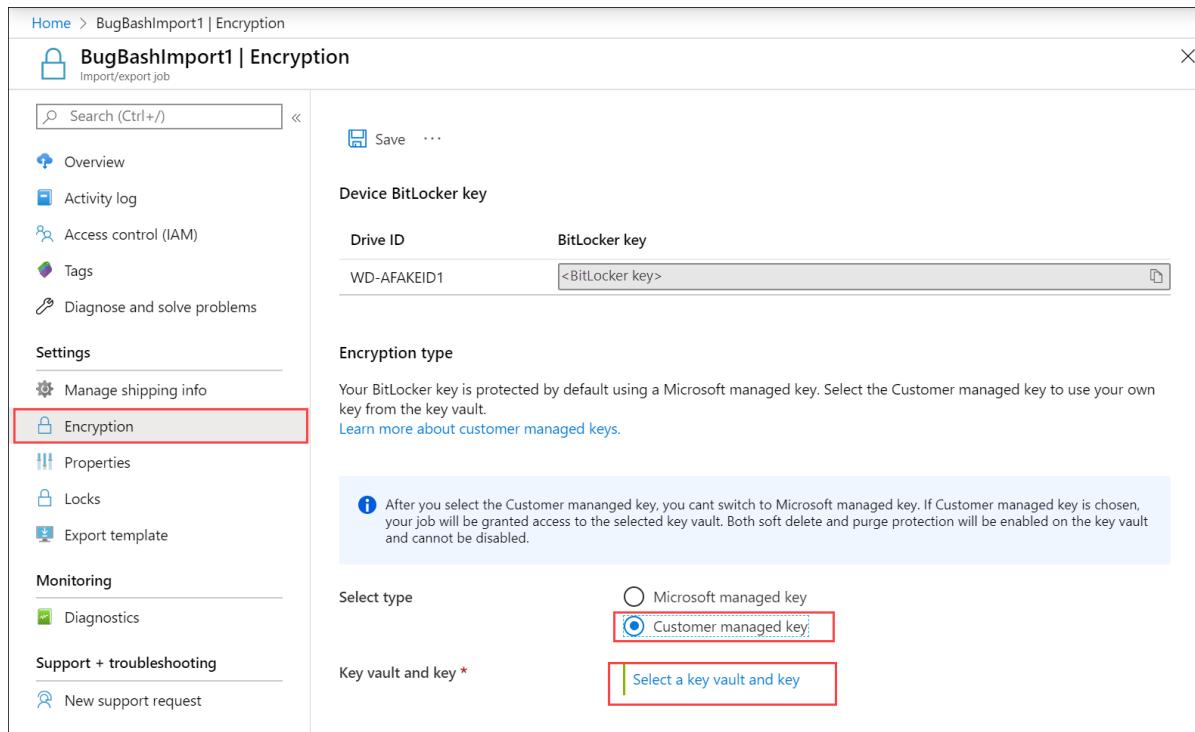
After you select the Customer managed key, you can't switch to Microsoft managed key. If Customer managed key is chosen, your job will be granted access to the selected key vault. Both soft delete and purge protection will be enabled on the key vault and cannot be disabled.

Select type

Microsoft managed key  
 Customer managed key

Key vault and key \*

Select a key vault and key



5. In the **Select key from Azure Key Vault** blade, the subscription is automatically populated. For **Key vault**, you can select an existing key vault from the dropdown list.

Home > BugBashImport1 | Encryption > Select key from Azure Key Vault

## Select key from Azure Key Vault

Subscription \*

<Subscription name>

Key vault \*

Create new

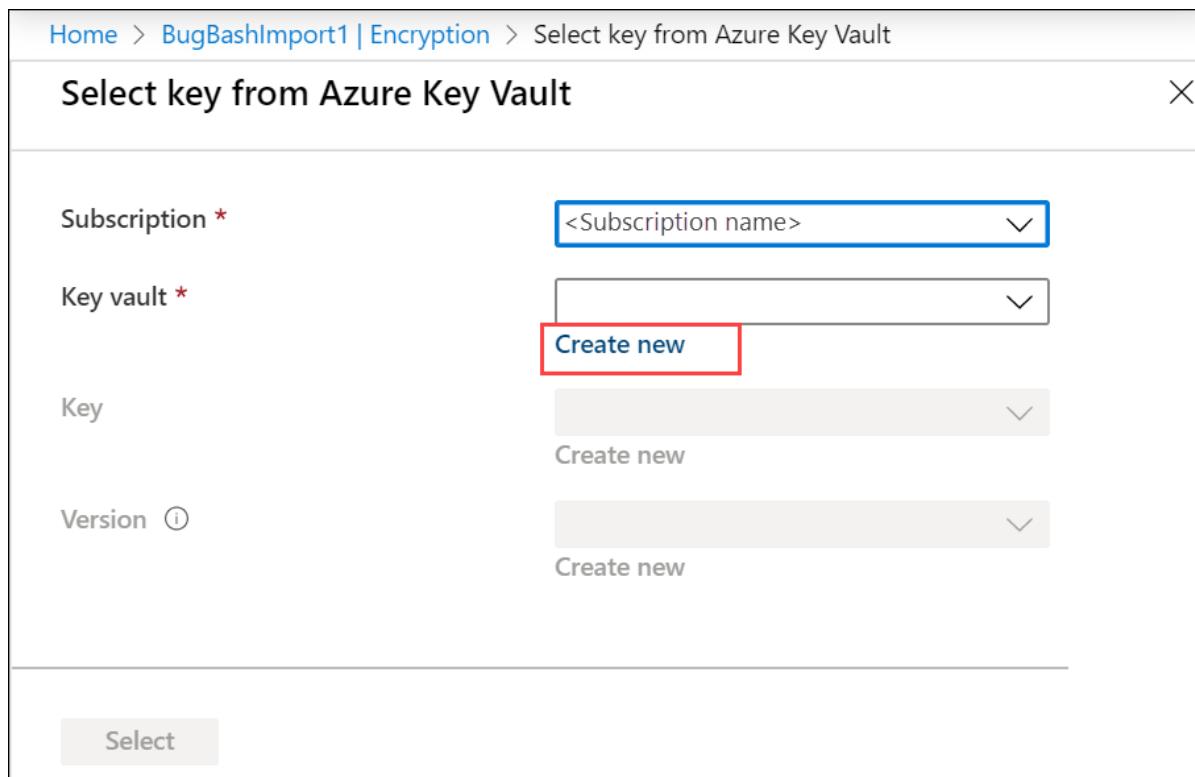
Key

Create new

Version ⓘ

Create new

Select



6. You can also select **Create new** to create a new key vault. In the **Create key vault blade**, enter the resource group and the key vault name. Accept all other defaults. Select **Review + Create**.

## Create key vault

Azure Key Vault is a cloud service used to manage keys, secrets, and certificates. Key Vault eliminates the need for developers to store security information in their code. It allows you to centralize the storage of your application secrets which greatly reduces the chances that secrets may be leaked. Key Vault also allows you to securely store secrets and keys backed by Hardware Security Modules or HSMs. The HSMs used are Federal Information Processing Standards (FIPS) 140-2 Level 2 validated. In addition, key vault provides logs of all access and usage attempts of your secrets so you have a complete audit trail for compliance. [Learn more](#)

Project details

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

Subscription <Subscription name>

Resource group \* CMK-BUGBASH-0503 [Create new](#)

Instance details

Key vault name \* iecmkky

Region (Asia Pacific) West India

Pricing tier \* Standard

Soft delete [Enable](#) [Disable](#)

Retention period (days) \* 90

Purge protection [Enable](#) [Disable](#)

[Review + create](#) [Next : Access policy >](#)

7. Review the information associated with your key vault and select **Create**. Wait for a couple minutes for the key vault creation to complete.

Home > BugBashImport1 | Encryption > Select key from Azure Key Vault > Create key vault

## Create key vault

Validation passed

Basics   Access policy   Networking   Tags   **Review + create**

**Basics**

Subscription	<Subscription name>
Resource group	CMK-BUGBASH-0503
Key vault name	iecmkkv
Region	West India
Pricing tier	Standard
Enable soft delete	Enabled
Enable purge protection	Disabled
Retention period (days)	90 days

**Access policy**

Azure Virtual Machines for deployment	Disabled
Azure Resource Manager for template deployment	Disabled
Azure Disk Encryption for volume encryption	Disabled
Azure Disk Encryption for volume encryption	Disabled
Permission model	Access control list
Access policies	1

**Networking**

Connectivity method	Public endpoint (all networks)
---------------------	--------------------------------

**Create**   < Previous   Next >   Download a template for automation

8. In the **Select key from Azure Key Vault**, you can select a key in the existing key vault.
9. If you created a new key vault, select **Create new** to create a key. RSA key size can be 2048 or greater.

### Select key from Azure Key Vault

Subscription \*

<Subscription name>

Key vault \*

iecmkkv

Create new

Key \*

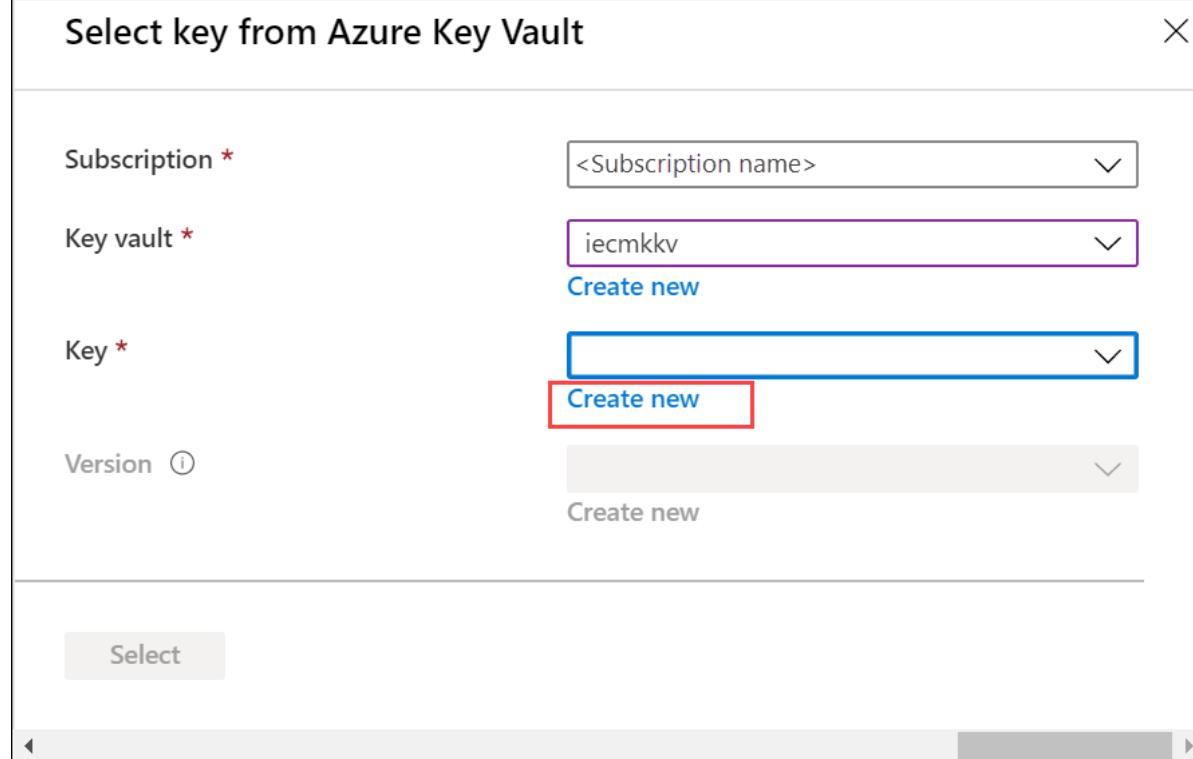
Create new

Version ⓘ

Create new

Select

◀ ▶



If the soft delete and purge protection are not enabled when you create the key vault, key vault will be updated to have soft delete and purge protection enabled.

10. Provide the name for your key, accept the other defaults, and select **Create**.

## Create a key

X

### Options

Generate ▾

Name \* ⓘ

iecmk ✓

Key Type ⓘ

RSA

EC

RSA Key Size

2048

3072

4096

Set activation date? ⓘ

Set expiration date? ⓘ

Enabled?

Yes

No

**Create**

11. Select the **Version** and then choose **Select**. You are notified that a key is created in your key vault.

## Select key from Azure Key Vault

**i** The key 'iecmk' has been successfully created.

Subscription *	<Subscription name>
Key vault *	iecmkkv
	<a href="#">Create new</a>
Key *	iecmk
	<a href="#">Create new</a>
Version * ⓘ	34565bafa244491b8af2c7749eb79a0b
	<a href="#">Create new</a>

**Select**

In the Encryption blade, you can see the key vault and the key selected for your customer managed key.

## Disable keys

You can only disable Microsoft managed keys and move to customer managed keys at any stage of the import/export job. However, you cannot disable the customer managed key once you have created it.

## Troubleshoot customer managed key errors

If you receive any errors related to your customer managed key, use the following table to troubleshoot:

Error code	Details	Recoverable?
CmkErrorAccessRevoked	Applied a customer managed key but the key access is currently revoked. For more information, see how to <a href="#">Enable the key access</a> .	Yes, check if: 1. Key vault still has the MSI in the access policy. 2. Access policy provides permissions to Get, Wrap, Unwrap. 3. If key vault is in a vNet behind the firewall, check if <b>Allow Microsoft Trusted Services</b> is enabled.
CmkErrorDisabled	Applied a customer managed key but the key is disabled. For more information, see how to <a href="#">Enable the key</a> .	Yes, by enabling the key version

Error code	Details	Recoverable?
CmkErrorNotFound	<p>Applied a customer managed key but can't find the key.</p> <p>If the key is deleted and purged after the retention period, you can't recover the key. If you backed up the key, you can restore the key to resolve this issue.</p>	<p>No, the key has been deleted and also got purged after the retention period.</p> <p>Yes, only if the customer has the key backed-up and restores it.</p>
CmkErrorVaultNotFound	<p>Applied a customer managed key but can't find the key vault associated with the key.</p> <p>If you deleted the key vault, you can't recover the customer managed key. If you migrated the key vault to a different tenant, see <a href="#">Change a key vault tenant ID after a subscription move</a>.</p>	<p>No, if the customer has deleted the key vault.</p> <p>Yes, if key vault underwent a tenant migration, then do one of:</p> <ol style="list-style-type: none"> <li>1. move back the key vault to the old tenant.</li> <li>2. set Identity = None and then back to Identity = SystemAssigned, this deletes and recreates the identity</li> </ol>

## Next steps

- [What is Azure Key Vault?](#)

# View the status of Azure Import/Export jobs

10/23/2019 • 4 minutes to read • [Edit Online](#)

This article provides information on how to view the drive and job status for Azure Import/Export jobs. Azure Import/Export service is used to securely transfer large amounts of data to Azure Blobs and Azure Files. The service is also used to export data from Azure Blob storage.

## View job and drive status

You can track the status of your import or export jobs from the Azure portal. Click the **Import/Export** tab. A list of your jobs appears on the page.

The screenshot shows the Azure Import/Export job interface. On the left, a sidebar lists various job management options like Overview, Activity log, Access control (IAM), Tags, and Diagnose and solve problems. The main area displays a job named 'import1'. At the top, there's a warning message: 'Update job status and tracking info once drives are shipped.' Below it, the 'Essentials' section shows the Resource group ('xtransport'), Status ('Shipping' is highlighted with a red box), Location ('West US'), Subscription name ('IaaSExp Dev'), and Subscription ID ('5c8a39f1-74d3-480d-9d9e-52f45977f29d'). To the right, detailed information for two drives is shown: Drive ID '14310CD5BC48' and Drive ID '28310CD5BC50', both in 'Specified' state with 0% complete. The right pane contains sections for Manifest information, MANIFEST FILE ('\\DriveManifest.xml'), HASH ('428AEB94976CEBCEC02BBFFC183FC35E'), URI ('-'), Log URI ('-'), VERBOSITY LOGS ('-'), and ERROR LOGS ('-').

## View job status

You see one of the following job statuses depending on where your drive is in the process.

JOB STATUS	DESCRIPTION
Creating	After a job is created, its state is set to <b>Creating</b> . While the job is in the <b>Creating</b> state, the Import/Export service assumes the drives have not been shipped to the data center. A job may remain in this state for up to two weeks, after which it is automatically deleted by the service.
Shipping	After you ship your package, you should update the tracking information in the Azure portal. This turns the job into <b>Shipping</b> state. The job remains in the <b>Shipping</b> state for up to two weeks.

JOB STATUS	DESCRIPTION
Received	After all drives are received at the data center, the job state is set to <b>Received</b> .
Transferring	Once at least one drive has begun processing, the job state is set to <b>Transferring</b> . For more information, go to <a href="#">Drive States</a> .
Packaging	After all drives have completed processing, the job is placed in <b>Packaging</b> state until the drives are shipped back to you.
Completed	After all drives are shipped back to you, if the job has completed without errors, then the job is set to <b>Completed</b> . The job is automatically deleted after 90 days in the <b>Completed</b> state.
Closed	After all drives are shipped back to you, if there were any errors during the processing of the job, the job is set to <b>Closed</b> . The job is automatically deleted after 90 days in the <b>Closed</b> state.

## View drive status

The table below describes the life cycle of an individual drive as it transitions through an import or export job. The current state of each drive in a job is seen in the Azure portal.

The following table describes each state that each drive in a job may pass through.

DRIVE STATE	DESCRIPTION
Specified	For an import job, when the job is created from the Azure portal, the initial state for a drive is <b>Specified</b> . For an export job, since no drive is specified when the job is created, the initial drive state is <b>Received</b> .
Received	The drive transitions to the <b>Received</b> state when the Import/Export service has processed the drives that were received from the shipping company for an import job. For an export job, the initial drive state is the <b>Received</b> state.
NeverReceived	The drive moves to the <b>NeverReceived</b> state when the package for a job arrives but the package doesn't contain the drive. A drive also moves into this state if it has been two weeks since the service received the shipping information, but the package has not yet arrived at the datacenter.
Transferring	A drive moves to the <b>Transferring</b> state when the service begins to transfer data from the drive to Azure Storage.
Completed	A drive moves to the <b>Completed</b> state when the service has successfully transferred all the data with no errors.

DRIVE STATE	DESCRIPTION
CompletedMoreInfo	A drive moves to the <b>CompletedMoreInfo</b> state when the service has encountered some issues while copying data either from or to the drive. The information can include errors, warnings, or informational messages about overwriting blobs.
ShippedBack	A drive moves to the <b>ShippedBack</b> state when it has been shipped from the datacenter back to the return address.

This image from the Azure portal displays the drive state of an example job:

DRIVE ID	DRIVE STATE	PERCENT COMPLETE
WD-WX51DA476H28	Completed with info	-
WD-WX21D5533HP2	Transferring	96%

The following table describes the drive failure states and the actions taken for each state.

DRIVE STATE	EVENT	RESOLUTION / NEXT STEP
NeverReceived	A drive that is marked as <b>NeverReceived</b> (because it was not received as part of the job's shipment) arrives in another shipment.	The operations team moves the drive to <b>Received</b> .
N/A	A drive that is not part of any job arrives at the datacenter as part of another job.	The drive is marked as an extra drive and is returned to you when the job associated with the original package is completed.

## Time to process job

The amount of time it takes to process an import/export job varies based on a number of factors such as:

- Shipping time
- Load at the datacenter

- Job type and size of the data being copied
- Number of disks in a job.

Import/Export service does not have an SLA but the service strives to complete the copy in 7 to 10 days after the disks are received. In addition to the status posted on Azure Portal, REST APIs can be used to track the job progress. The percent complete parameter in the [List Jobs](#) operation API call provides the percentage copy progress.

## Next steps

- [Set up the WAImportExport tool](#)
- [Transfer data with AzCopy command-line utility](#)
- [Azure Import Export REST API sample](#)

# Reviewing Azure Import/Export job status with copy log files

1/14/2020 • 2 minutes to read • [Edit Online](#)

When the Microsoft Azure Import/Export service processes drives associated with an import or export job, it writes copy log files to the storage account to or from which you are importing or exporting blobs. The log file contains detailed status about each file that was imported or exported. The URL to each copy log file is returned when you query the status of a completed job; see [Get Job](#) for more information.

## Example URLs

The following are example URLs for copy log files for an import job with two drives:

`http://myaccount.blob.core.windows.net/ImportExportStatesPath/waies/myjob_9WM35C2V_20130921-034307-902_error.xml`

`http://myaccount.blob.core.windows.net/ImportExportStatesPath/waies/myjob_9WM45A6Q_20130921-042122-021_error.xml`

See [Import/Export service Log File Format](#) for the format of copy logs and the full list of status codes.

## Next steps

- [Setting Up the Azure Import/Export Tool](#)
- [Preparing hard drives for an import job](#)
- [Repairing an import job](#)
- [Repairing an export job](#)
- [Troubleshooting the Azure Import/Export Tool](#)

# Repairing an import job

12/9/2019 • 4 minutes to read • [Edit Online](#)

The Microsoft Azure Import/Export service may fail to copy some of your files or parts of a file to the Windows Azure Blob service. Some reasons for failures include:

- Corrupted files
- Damaged drives
- The storage account key changed while the file was being transferred.

You can run the Microsoft Azure Import/Export Tool with the import job's copy log files, and the tool uploads the missing files (or parts of a file) to your Windows Azure storage account to complete the import job.

## RepairImport parameters

The following parameters can be specified with **RepairImport**:

/r:<RepairFile>	<b>Required.</b> Path to the repair file, which tracks the progress of the repair, and allows you to resume an interrupted repair. Each drive must have one and only one repair file. When you start a repair for a given drive, pass in the path to a repair file, which does not yet exist. To resume an interrupted repair, you should pass in the name of an existing repair file. The repair file corresponding to the target drive must always be specified.
/logdir:<LogDirectory>	<b>Optional.</b> The log directory. Verbose log files are written to this directory. If no log directory is specified, the current directory is used as the log directory.
/d:<TargetDirectories>	<b>Required.</b> One or more semicolon-separated directories that contain the original files that were imported. The import drive may also be used, but is not required if alternate locations of original files are available.
/bk:<BitLockerKey>	<b>Optional.</b> You should specify the BitLocker key if you want the tool to unlock an encrypted drive where the original files are available.
/sn:<StorageAccountName>	<b>Required.</b> The name of the storage account for the import job.
/sk:<StorageAccountKey>	<b>Required</b> if and only if a container SAS is not specified. The account key for the storage account for the import job.
/csas:<ContainerSas>	<b>Required</b> if and only if the storage account key is not specified. The container SAS for accessing the blobs associated with the import job.

/CopyLogFile:<DriveCopyLogFile>	<b>Required.</b> Path to the drive copy log file (either verbose log or error log). The file is generated by the Windows Azure Import/Export service and can be downloaded from the blob storage associated with the job. The copy log file contains information about failed blobs or files, which are to be repaired.
/PathMapFile:<DrivePathMapFile>	<b>Optional.</b> Path to a text file that can be used to resolve ambiguities if you have multiple files with the same name that you were importing in the same job. The first time the tool is run, it can populate this file with all of the ambiguous names. Subsequent runs of the tool use this file to resolve the ambiguities.

## Using the RepairImport command

To repair import data by streaming the data over the network, you must specify the directories that contain the original files you were importing using the `/d` parameter. You must also specify the copy log file that you downloaded from your storage account. A typical command line to repair an import job with partial failures looks like:

```
WAIImportExport.exe RepairImport /r:C:\WAIImportExport\9WM35C2V.rep /d:C:\Users\bob\Pictures;X:\BobBackup\photos
/sn:bobmediaaccount
/sk:VkGbrUqBWLYJ6zg1m29VOTrxpBgdN0lp+kp0C9MEDx3GELxmBw4hK94f7KysbbeKLdksg7VoN1W/a5UuM2zNgQ==
/CopyLogFile:C:\WAIImportExport\9WM35C2V.log
```

In the following example of a copy log file, one 64-K piece of a file was corrupted on the drive that was shipped for the import job. Since this is the only failure indicated, the rest of the blobs in the job were successfully imported.

```
<?xml version="1.0" encoding="utf-8"?>
<DriveLog>
<DriveId>9WM35C2V</DriveId>
<Blob Status="CompletedWithErrors">
<BlobPath>pictures/animals/koala.jpg</BlobPath>
<FilePath>\animals\koala.jpg</FilePath>
<Length>163840</Length>
<ImportDisposition Status="Overwritten">overwrite</ImportDisposition>
<PageRangeList>
<PageRange Offset="65536" Length="65536" Hash="AA2585F6F6FD01C4AD4256E018240CD4" Status="Corrupted" />
</PageRangeList>
</Blob>
<Status>CompletedWithErrors</Status>
</DriveLog>
```

When this copy log is passed to the Azure Import/Export Tool, the tool tries to finish the import for this file by copying the missing contents across the network. Following the example above, the tool looks for the original file `\animals\koala.jpg` within the two directories `C:\Users\bob\Pictures` and `X:\BobBackup\photos`. If the file `C:\Users\bob\Pictures\animals\koala.jpg` exists, the Azure Import/Export Tool copies the missing range of data to the corresponding blob `http://bobmediaaccount.blob.core.windows.net/pictures/animals/koala.jpg`.

## Resolving conflicts when using RepairImport

In some situations, the tool may not be able to find or open the necessary file for one of the following reasons: the file could not be found, the file is not accessible, the file name is ambiguous, or the content of the file is no longer correct.

An ambiguous error could occur if the tool is trying to locate `\animals\koala.jpg` and there is a file with that name under both `C:\Users\bob\pictures` and `X:\BobBackup\photos`. That is, both `C:\Users\bob\pictures\animals\koala.jpg` and `X:\BobBackup\photos\animals\koala.jpg` exist on the import job drives.

The `/PathMapFile` option allows you to resolve these errors. You can specify the name of the file, which contains the list of files that the tool was not able to correctly identify. The following command-line example populates `9WM35C2V_pathmap.txt`:

```
WAImportExport.exe RepairImport /r:C:\WAImportExport\9WM35C2V.rep /d:C:\Users\bob\Pictures;X:\BobBackup\photos
/sn:bobmediaaccount
/sk:VkGbrUqBWLYJ6zg1m29VOTrxpBgdN0lp+kp0C9MEDx3GELxmBw4hK94f7KysbbeKLDksg7VoN1W/a5UuM2zNgQ==
/CopyLogFile:C:\WAImportExport\9WM35C2V.log /PathMapFile:C:\WAImportExport\9WM35C2V_pathmap.txt
```

The tool will then write the problematic file paths to `9WM35C2V_pathmap.txt`, one on each line. For instance, the file may contain the following entries after running the command:

```
\animals\koala.jpg
\animals\kangaroo.jpg
```

For each file in the list, you should attempt to locate and open the file to ensure it is available to the tool. If you wish to tell the tool explicitly where to find a file, you can modify the path map file and add the path to each file on the same line, separated by a tab character:

```
\animals\koala.jpg C:\Users\bob\Pictures\animals\koala.jpg
\animals\kangaroo.jpg X:\BobBackup\photos\animals\kangaroo.jpg
```

After making the necessary files available to the tool, or updating the path map file, you can rerun the tool to complete the import process.

## Next steps

- [Setting Up the Azure Import/Export Tool](#)
- [Preparing hard drives for an import job](#)
- [Reviewing job status with copy log files](#)
- [Repairing an export job](#)
- [Troubleshooting the Azure Import/Export Tool](#)

# Repairing an export job

12/9/2019 • 5 minutes to read • [Edit Online](#)

After an export job has completed, you can run the Microsoft Azure Import/Export Tool on-premises to:

1. Download any files that the Azure Import/Export service was unable to export.
2. Validate that the files on the drive were correctly exported.

You must have connectivity to Azure Storage to use this functionality.

The command for repairing an import job is **RepairExport**.

## RepairExport parameters

The following parameters can be specified with **RepairExport**:

PARAMETER	DESCRIPTION
/r:<RepairFile>	Required. Path to the repair file, which tracks the progress of the repair, and allows you to resume an interrupted repair. Each drive must have one and only one repair file. When you start a repair for a given drive, you will pass in the path to a repair file which does not yet exist. To resume an interrupted repair, you should pass in the name of an existing repair file. The repair file corresponding to the target drive must always be specified.
/logdir:<LogDirectory>	Optional. The log directory. Verbose log files will be written to this directory. If no log directory is specified, the current directory will be used as the log directory.
/d:<TargetDirectory>	Required. The directory to validate and repair. This is usually the root directory of the export drive, but could also be a network file share containing a copy of the exported files.
/bk:<BitLockerKey>	Optional. You should specify the BitLocker key if you want the tool to unlock an encrypted where the exported files are stored.
/sn:<StorageAccountName>	Required. The name of the storage account for the export job.
/sk:<StorageAccountKey>	<b>Required</b> if and only if a container SAS is not specified. The account key for the storage account for the export job.
/csas:<ContainerSas>	<b>Required</b> if and only if the storage account key is not specified. The container SAS for accessing the blobs associated with the export job.
/CopyLogFile:<DriveCopyLogFile>	Required. The path to the drive copy log file. The file is generated by the Windows Azure Import/Export service and can be downloaded from the blob storage associated with the job. The copy log file contains information about failed blobs or files which are to be repaired.

PARAMETER	DESCRIPTION
/ManifestFile:<DriveManifestFile>	<p>Optional. The path to the export drive's manifest file. This file is generated by the Windows Azure Import/Export service and stored on the export drive, and optionally in a blob in the storage account associated with the job.</p> <p>The content of the files on the export drive will be verified with the MD5 hashes contained in this file. Any files that are determined to be corrupted will be downloaded and rewritten to the target directories.</p>

## Using RepairExport mode to correct failed exports

You can use the Azure Import/Export Tool to download files that failed to export. The copy log file will contain a list of files that failed to export.

The causes of export failures include the following possibilities:

- Damaged drives
- The storage account key changed during the transfer process

To run the tool in **RepairExport** mode, you first need to connect the drive containing the exported files to your computer. Next, run the Azure Import/Export Tool, specifying the path to that drive with the `/d` parameter. You also need to specify the path to the drive's copy log file that you downloaded. The following command line example below runs the tool to repair any files that failed to export:

```
WAImportExport.exe RepairExport /r:C:\WAImportExport\9WM35C3U.rep /d:G:\ /sn:bobmediaaccount
/sk:VkGbrUqBWLYJ6zg1m29VOTrxpBgdN0lp+kp0C9MEdx3GELxmBw4hK94f7KysbbeKLdksg7VoN1W/a5UuM2zNgQ==
/CopyLogFile:C:\WAImportExport\9WM35C3U.log
```

The following is an example of a copy log file that shows that one block in the blob failed to export:

```
<?xml version="1.0" encoding="utf-8"?>
<DriveLog>
 <DriveId>9WM35C2V</DriveId>
 <Blob Status="CompletedWithErrors">
 <BlobPath>pictures/wild/desert.jpg</BlobPath>
 <FilePath>\pictures\wild\desert.jpg</FilePath>
 <LastModified>2012-09-18T23:47:08Z</LastModified>
 <Length>163840</Length>
 <BlockList>
 <Block Offset="65536" Length="65536" Id="AQAAAA==" Status="Failed" />
 </BlockList>
 </Blob>
 <Status>CompletedWithErrors</Status>
</DriveLog>
```

The copy log file indicates that a failure occurred while the Windows Azure Import/Export service was downloading one of the blob's blocks to the file on the export drive. The other components of the file downloaded successfully, and the file length was correctly set. In this case, the tool will open the file on the drive, download the block from the storage account, and write it to the file range starting from offset 65536 with length 65536.

## Using RepairExport to validate drive contents

You can also use Azure Import/Export with the **RepairExport** option to validate the contents on the drive are correct. The manifest file on each export drive contains MD5s for the contents of the drive.

The Azure Import/Export service can also save the manifest files to a storage account during the export process.

The location of the manifest files is available via the [Get Job](#) operation when the job has completed. See

[Import/Export service Manifest File Format](#) for more information about the format of a drive manifest file.

The following example shows how to run the Azure Import/Export Tool with the **/ManifestFile** and **/CopyLogFile** parameters:

```
WAImportExport.exe RepairExport /r:C:\WAImportExport\9WM35C3U.rep /d:G:\ /sn:bobmediaaccount
/sk:VkGbrUqBWLYJ6zg1m29VOTrxpBgdN0lp+kp0C9MEdx3GELxmBw4hK94f7KysbbeKLDksg7VoN1W/a5UuM2zNgQ==
/CopyLogFile:C:\WAImportExport\9WM35C3U.log /ManifestFile:G:\9WM35C3U.manifest
```

The following is an example of a manifest file:

```
<?xml version="1.0" encoding="utf-8"?>
<DriveManifest Version="2011-10-01">
 <Drive>
 <DriveId>9WM35C3U</DriveId>
 <ClientCreator>Windows Azure Import/Export service</ClientCreator>
 <BlobList>
 <Blob>
 <BlobPath>pictures/city/redmond.jpg</BlobPath>
 <FilePath>\pictures\city\redmond.jpg</FilePath>
 <Length>15360</Length>
 <PageRangeList>
 <PageRange Offset="0" Length="3584" Hash="72FC55ED9AFDD40A0C8D5C4193208416" />
 <PageRange Offset="3584" Length="3584" Hash="68B28A561B73D1DA769D4C24AA427DB8" />
 <PageRange Offset="7168" Length="512" Hash="F521DF2F50C46BC5F9EA9FB787A23EED" />
 </PageRangeList>
 <PropertiesPath
 Hash="E72A22EA959566066AD89E3B49020C0A">\pictures\city\redmond.jpg.properties</PropertiesPath>
 </Blob>
 <Blob>
 <BlobPath>pictures/wild/canyon.jpg</BlobPath>
 <FilePath>\pictures\wild\canyon.jpg</FilePath>
 <Length>10884</Length>
 <BlockList>
 <Block Offset="0" Length="2721" Id="AAAAAA==" Hash="263DC9C4B99C2177769C5EBE04787037" />
 <Block Offset="2721" Length="2721" Id="AQAAA==" Hash="0C52BAE2CC20EFEC15CC1E3045517AA6" />
 <Block Offset="5442" Length="2721" Id="AgAAA==" Hash="73D1CB62CB426230C34C9F57B7148F10" />
 <Block Offset="8163" Length="2721" Id="AwAAA==" Hash="11210E665C5F8E7E4F136D053B243E6A" />
 </BlockList>
 <PropertiesPath
 Hash="81D7F81B2C29F10D6E123D386C3A4D5A">\pictures\wild\canyon.jpg.properties</PropertiesPath>
 </Blob>
 </BlobList>
 </Drive>
</DriveManifest>
```

After finishing the repair process, the tool will read through each file referenced in the manifest file and verify the file's integrity with the MD5 hashes. For the manifest above, it will go through the following components.

```
G:\pictures\city\redmond.jpg, offset 0, length 3584
G:\pictures\city\redmond.jpg, offset 3584, length 3584
G:\pictures\city\redmond.jpg, offset 7168, length 3584
G:\pictures\city\redmond.jpg.properties
G:\pictures\wild\canyon.jpg, offset 0, length 2721
G:\pictures\wild\canyon.jpg, offset 2721, length 2721
G:\pictures\wild\canyon.jpg, offset 5442, length 2721
G:\pictures\wild\canyon.jpg, offset 8163, length 2721
G:\pictures\wild\canyon.jpg.properties
```

Any component failing the verification will be downloaded by the tool and rewritten to the same file on the drive.

## Next steps

- [Setting Up the Azure Import/Export Tool](#)
- [Preparing hard drives for an import job](#)
- [Reviewing job status with copy log files](#)
- [Repairing an import job](#)
- [Troubleshooting the Azure Import/Export Tool](#)

# Azure Import/Export service: frequently asked questions

4/1/2020 • 6 minutes to read • [Edit Online](#)

The following are questions and answers that you may have when you use your Azure Import/Export service to transfer data into Azure storage. Questions and answers are arranged in the following categories:

- About Import/Export service
- Preparing the disks for import/export
- Import/Export jobs
- Shipping disks
- Miscellaneous

## About Import/Export service

### **Can I copy Azure File storage using the Azure Import/Export service?**

Yes. The Azure Import/Export service supports import into Azure File Storage. It does not support export of Azure Files at this time.

### **Is the Azure Import/Export service available for CSP subscriptions?**

Yes. Azure Import/Export service supports Cloud Solution Providers (CSP) subscriptions.

### **Can I use the Azure Import/Export service to copy PST mailboxes and SharePoint data to O365?**

Yes. For more information, go to [Import PST files or SharePoint data to Office 365](#).

### **Can I use the Azure Import/Export service to copy my backups offline to the Azure Backup Service?**

Yes. For more information, go to [Offline Backup workflow in Azure Backup](#).

### **Can I purchase drives for import/export jobs from Microsoft?**

No. You need to ship your own drives for import and export jobs.

## Preparing disks for import/export

### **Can I skip the drive preparation step for an import job? Can I prepare a drive without copying?**

No. Any drive used to import data must be prepared using the Azure WAImportExport tool. Use the tool to also copy data to the drive.

### **Do I need to perform any disk preparation when creating an export job?**

No. Some prechecks are recommended. To check the number of disks required, use the WAImportExport tool's PreviewExport command. For more information, see [Previewing Drive Usage for an Export Job](#). The command helps you preview drive usage for the selected blobs, based on the size of the drives you are going to use. Also check that you can read from and write to the hard drive that is shipped for the export job.

## Import/Export jobs

### **Can I cancel my job?**

Yes. You can cancel a job when its status is **Creating** or **Shipping**. Beyond these stages, job cannot be canceled and it continues until the final stage.

## **How long can I view the status of completed jobs in the Azure portal?**

You can view the status for completed jobs for up to 90 days. Completed jobs are deleted after 90 days.

## **If I want to import or export more than 10 drives, what should I do?**

One import or export job can reference only 10 drives in a single job. To ship more than 10 drives, you should create multiple jobs. Drives associated with the same job must be shipped together in the same package. For more information and guidance when data capacity spans multiple disk import jobs, contact Microsoft Support.

## **The uploaded blob shows status as "Lease expired". What should I do?**

You can ignore the "Lease Expired" field. Import/Export takes lease on the blob during upload to make sure that no other process can update the blob in parallel. Lease Expired implies that Import/export is no longer uploading to it and the blob is available for your use.

# Shipping disks

## **What is the maximum number of HDD for in one shipment?**

There is no limit to the number of HDDs in one shipment. If the disks belong to multiple jobs, we recommend that you:

- label the disks with corresponding job names.
- update the jobs with a tracking number suffixed with -1, -2 etc.

## **Should I include anything other than the HDD in my package?**

Ship only your hard drives in the shipping package. Do not include items like power supply cables or USB cables.

## **Do I have to ship my drives using FedEx or DHL?**

You can ship drives to the Azure datacenter using any known carrier like FedEx, DHL, UPS, or US Postal Service. However, for return shipment of drives to you from the datacenter, you must provide:

- A FedEx account number in the US and EU, or
- A DHL account number in the Asia and Australia regions.

### **NOTE**

The datacenters in India require a declaration letter on your letterhead (delivery challan) to return the drives. To arrange the required entry pass, you must also book the pick up with your selected carrier and share the details with the datacenter.

## **Are there any restrictions with shipping and returning my drive internationally?**

Please note that the physical media that you are shipping may need to cross international borders. You are responsible for ensuring that your physical media and data are imported and/or exported in accordance with the applicable laws. Before shipping the physical media, check with your advisors to verify that your media and data can legally be shipped to the identified data center. This will help to ensure that it reaches Microsoft in a timely manner.

After the upload is complete, the process to return drive(s) to an international address can take longer than the typical 2-3 days needed for local shipping. During the stage listed in the Azure portal as Packaging, the Data Box team is ensuring that the correct documentation is provided to ensure the shipment complies with the various international import and export requirements.

## **Are there any special requirements for delivering my disks to a datacenter?**

The requirements depend on the specific Azure datacenter restrictions.

- There are a few sites, like Australia, Germany, and UK South, that require a Microsoft datacenter Inbound ID number to be written on the parcel for security reasons. Before you ship your drives or disks to the datacenter,

contact Azure DataBox Operations ([adbops@microsoft.com](mailto:adbops@microsoft.com)) to get this number. Without this number, the package will be rejected.

- The datacenters in India require the personal details of the driver, such as the Government ID Card or Proof No. (for example, PAN, AADHAR, DL), name, contact, and the car plate number to get a gate entry pass. To avoid delivery delays, inform your carrier about these requirements.

### **When creating a job, the shipping address is a location that is different from my storage account location. What should I do?**

Some storage account locations are mapped to alternate shipping locations. Previously available shipping locations can also be temporarily mapped to alternate locations. Always check the shipping address provided during job creation before shipping your drives.

### **When shipping my drive, the carrier asks for the data center contact address and phone number. What should I provide?**

The phone number and DC address is provided as part of job creation.

## Miscellaneous

### **What happens if I accidentally send an HDD that does not conform to the supported requirements?**

The Azure data center will return the drive that does not conform to the supported requirements to you. If only some of the drives in the package meet the support requirements, those drives will be processed, and the drives that do not meet the requirements will be returned to you.

### **Does the service format the drives before returning them?**

No. All drives are encrypted with BitLocker.

### **How can I access data that is imported by this service?**

Use the Azure portal or [Storage Explorer](#) to access the data under your Azure storage account.

### **After the import is complete, what does my data look like in the storage account? Is my directory hierarchy preserved?**

When preparing a hard drive for an import job, the destination is specified by the DstBlobPathOrPrefix field in the dataset CSV. This is the destination container in the storage account to which data from the hard drive is copied. Within this destination container, virtual directories are created for folders from the hard drive and blobs are created for files.

### **If a drive has files that already exist in my storage account, does the service overwrite existing blobs or files?**

Depends. When preparing the drive, you can specify whether the destination files should be overwritten or ignored using the field in dataset CSV file called Disposition:<rename|no-overwrite|overwrite>. By default, the service renames the new files rather than overwrite existing blobs or files.

### **Is the WAImportExport tool compatible with 32-bit operating systems?**

No. The WAImportExport tool is only compatible with 64-bit Windows operating systems. For a complete list of Supported OS, go to [Supported Operating Systems](#).

### **What is the maximum Block Blob and Page Blob Size supported by Azure Import/Export?**

- Max Block Blob size is approximately 4.768TB or 5,000,000 MB.
- Max Page Blob size is 8TB.

### **Does Azure Import/Export support AES-256 encryption?**

No. Azure Import/Export service uses AES-128 BitLocker encryption.

## Next steps

- [What is Azure Import/Export?](#)



# Open a support ticket for an Import/Export job

7/7/2019 • 2 minutes to read • [Edit Online](#)

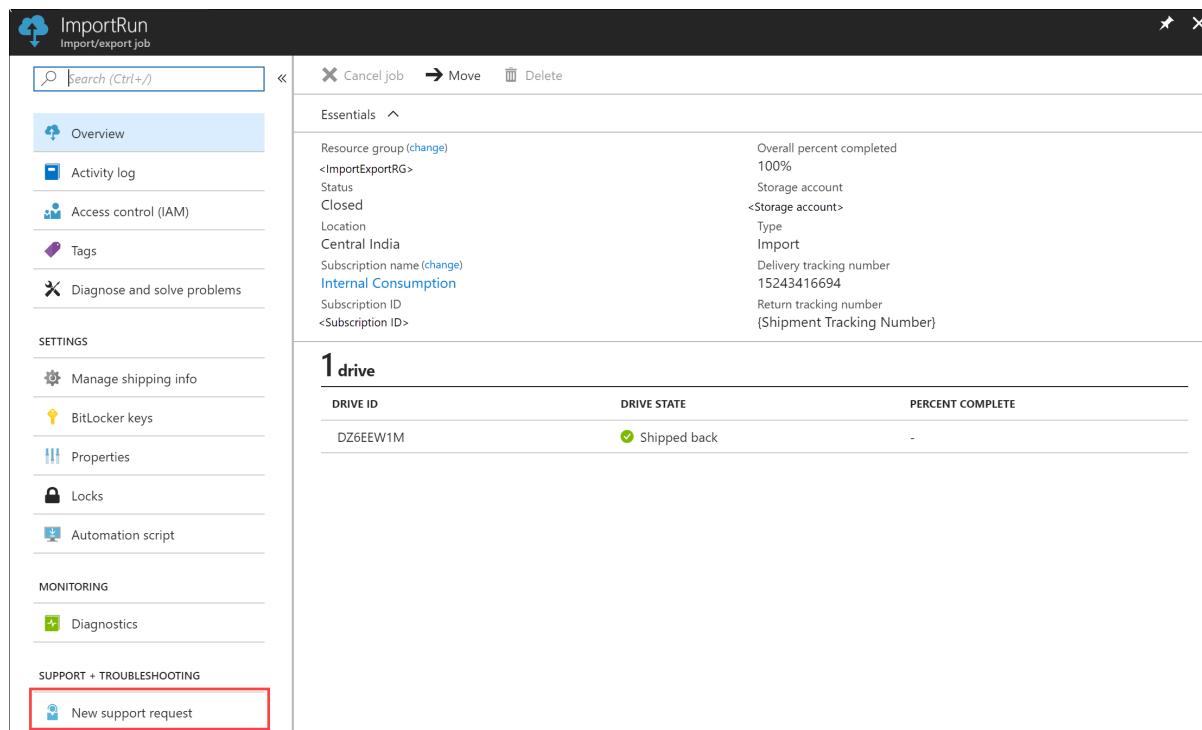
If you encounter any issues with your Import/Export service, you can create a service request for technical support. This article walks you through:

- How to create a support request.
- How to manage a support request lifecycle from within the portal.

## Create a support request

Perform the following steps to create a support request:

1. Go to your import/export job. Navigate to **SUPPORT + TROUBLESHOOTING** section and then click **New support request**.



The screenshot shows the Azure ImportRun portal interface. On the left, there's a navigation sidebar with sections like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, SETTINGS (Manage shipping info, BitLocker keys, Properties, Locks, Automation script), MONITORING (Diagnostics), and SUPPORT + TROUBLESHOOTING (New support request). The main area displays details for an import job, including Resource group, Status (Closed), Location (Central India), Subscription name (Internal Consumption), and various tracking numbers. Below this, there's a table for drives, showing one drive named DZ6EEW1M in a Shipped back state. The 'New support request' button in the sidebar is highlighted with a red box.

2. In **New support request**, select **Basics**. In **Basics**, do the following steps:
  - a. From the **Issue type** dropdown list, select **Technical**.
  - b. Choose your **Subscription**.
  - c. Under **Service**, check **My Services**. From the dropdown list, you can select one of the options - **Storage Account Management, Blob, or File**.
    - If you choose **Storage Account Management**, select **Resource**, and **Support plan**.

New support request

HELP + SUPPORT

1 Basics >

2 Problem >

3 Contact information >

Basics

NEW SUPPORT REQUEST

\* Issue type  
Technical

\* Subscription  
Internal Consumption

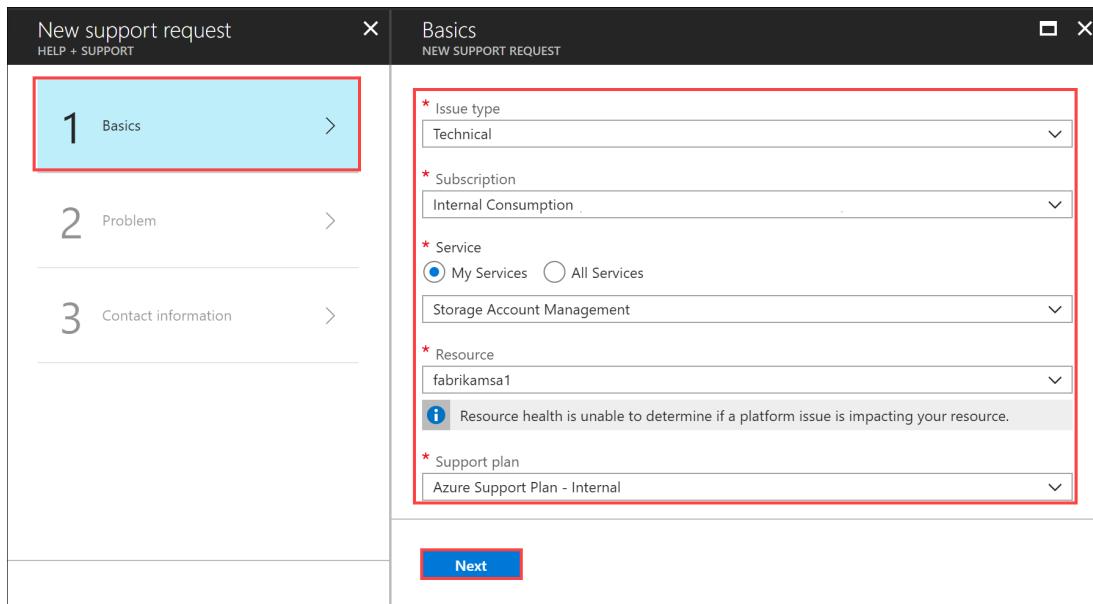
\* Service  
 My Services  All Services  
Storage Account Management

\* Resource  
fabrikamsa1

Resource health is unable to determine if a platform issue is impacting your resource.

\* Support plan  
Azure Support Plan - Internal

Next



- If you choose **Blob**, select **Resource**, **Container names** (optional), and **Support plan**.

New support request

HELP + SUPPORT

1 Basics >

2 Problem >

3 Contact information >

Basics

NEW SUPPORT REQUEST

\* Issue type  
Technical

\* Subscription  
Internal Consumption

\* Service  
 My Services  All Services  
Blob

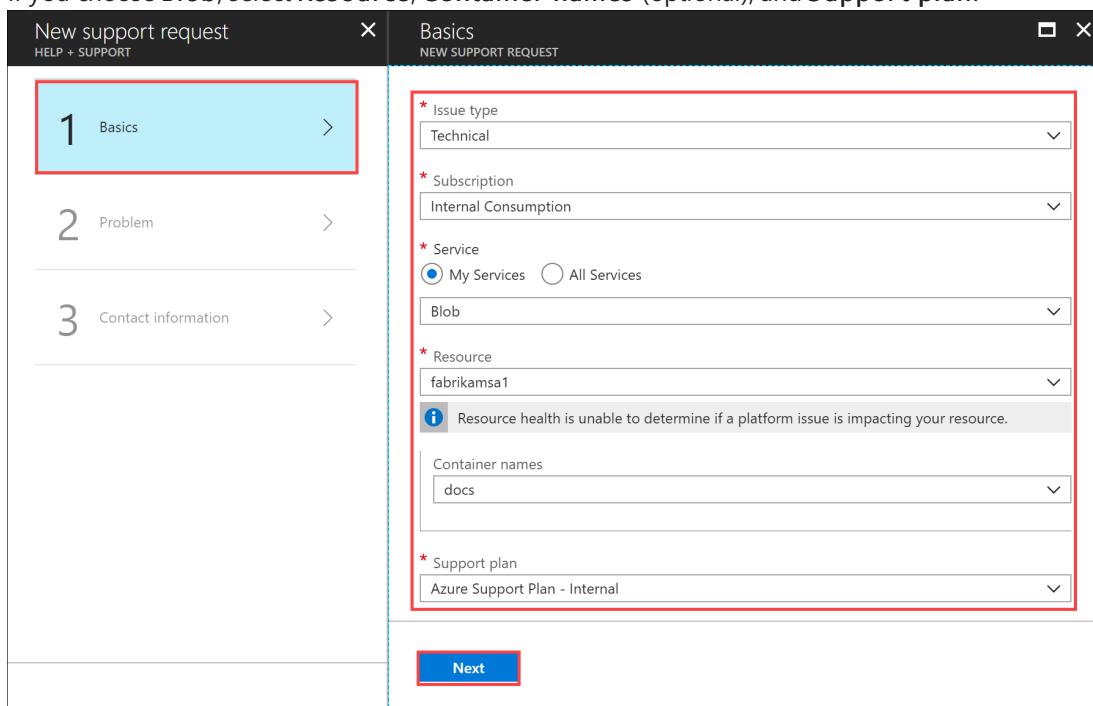
\* Resource  
fabrikamsa1

Resource health is unable to determine if a platform issue is impacting your resource.

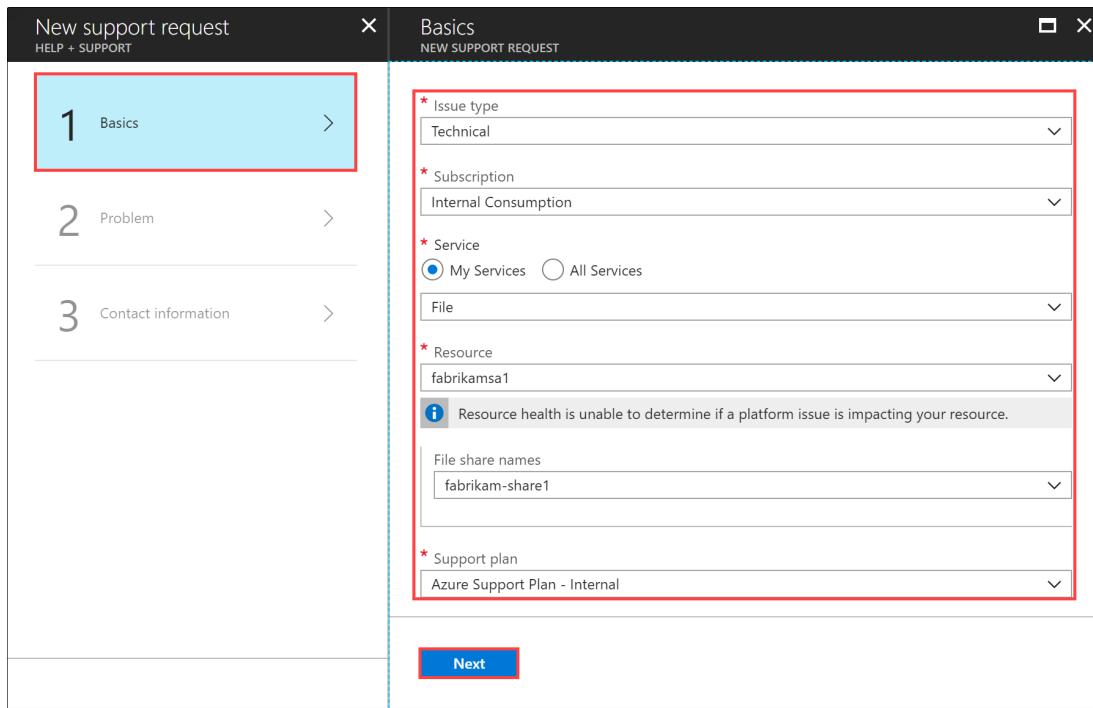
Container names  
docs

\* Support plan  
Azure Support Plan - Internal

Next



- If you choose **File**, select **Resource**, **File share names** (optional), and **Support plan**



- d. Click **Next**.
3. In **New support request**, select **Step 2 Problem**. In **Problem**, do the following steps:
- Choose the **Severity** as **C - Minimal impact**. Support will update it if needed.
  - Select the **Problem type** as **Data Migration**.
  - Choose the **Category** as **Import - Export**.
  - Provide a **Title** for the issue and more **Details**.
  - Provide the start date and time for the problem.
  - In the **File upload**, click the folder icon to browse any other files you want to upload.
  - Check **Share diagnostic information**.
  - Click **Next**.

The screenshot shows the Microsoft Support Center interface for creating a new support request. On the left, a vertical navigation bar lists three steps: 1 Basics (marked with a green checkmark), 2 Problem (highlighted with a red box), and 3 Contact information (marked with a green checkmark). The main panel is titled "Problem NEW SUPPORT REQUEST". It contains several input fields with validation feedback:

- Severity**: C - Minimal impact (green checkmark)
- Problem type**: Data migration (green checkmark)
- Category**: Import - Export (green checkmark)
- Title**: Can't find the status of the disk (green checkmark)
- Details**: Need to know the status of the disk import job for this blob. (green checkmark)

A note indicates: "Found solutions based on your problem description. See solutions on the right." Below the form, there are fields for "When did the problem start?", a file upload field, and a checkbox for "Share diagnostic information". A large red "Next" button is at the bottom.

4. In **New support request**, click **Step 3 Contact information**. In **Contact information**, do the following steps:
  - a. In the **Contact options**, provide your preferred contact method (phone or email) and the language. The response time is automatically selected based on your subscription plan.
  - b. In the Contact information, provide your name, email, optional contact, country/region. Select the **Save contact changes for future support requests** checkbox.
  - c. Click **Create**.

New support request

HELP + SUPPORT

**Contact options**

- \* Preferred contact method ⓘ

**Response ⓘ**

- 

**Language**

- 

**Contact information**

- \* First name
- \* Last name
- \* Email
- Who else should we email? ⓘ
- Phone number
- \* Country/region
- Save contact changes for future support requests.

By clicking create you accept the [terms and conditions](#). View our [privacy policy](#).

**Create**

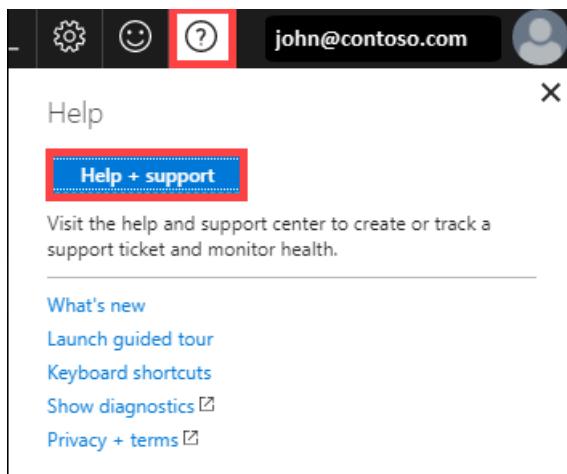
Microsoft Support will use this information to reach out to you for further information, diagnosis, and resolution. After you have submitted your request, a Support engineer will contact you as soon as possible to proceed with your request.

## Manage a support request

After creating a support ticket, you can manage the lifecycle of the ticket from within the portal.

### To manage your support requests

1. To get to the help and support page, navigate to **Browse > Help + support**.



2. A tabular listing of **Recent support requests** is displayed in **Help + support**.

The screenshot shows the Azure Help + support interface. On the left, there's a sidebar with links for Overview, New support request, All support requests, Support plans, Service issues, Planned maintenance, Health advisories, Health history, Resource health, Advisor, and Get started with Azure. The main area has a search bar and a section titled 'Have you tried one of these?' with four cards: 'Get started' (Azure's most-used features), 'Documentation' (Azure tutorials and how-to articles), 'Learn about billing' (tips for monitoring usage and understanding your bill), and 'Support plans' (choose the right Azure support plan). Below this is a 'Community' section with links to MSDN Forums, Stackoverflow, @AzureSupport on Twitter, Serverfault, and Azure on Serverfault. A 'Recent support requests' table follows, showing four entries:

TITLE	ID	CREATED (UTC)	SUBSCRIPTION	RESOURCE TYPE	UPDATED	STATUS
Enable West US 2 cloud services deployment f...	118031617830507	Fri, Mar 16, 2018, 6:05:54 ...	Email Platform Prod	Cloud Services (Web roles...)	11 days ago	Closed
Enable West US2 for this subscription and mo...	118031617830495	Fri, Mar 16, 2018, 6:04:44 ...	Email Platform Prod	Cloud Services (Web roles...)	8 days ago	Closed
Increase partition count for this Event hub to 1...	118031517824013	Thu, Mar 15, 2018, 10:04:5...	Email Platform Prod	Event Hubs	12 days ago	Closed
Can't find the status of the disk.	118031217797609	Mon, Mar 12, 2018, 9:06:3...	Internal Consumption	Storage Account Manage...	15 days ago	Open

A red box highlights the last row. At the bottom left of the table area, there's a link 'See all support requests'.

3. Select and click a support request. You can view the status and the details for this request. Click **+ New message** if you want to follow up on this request.

The screenshot shows a support request window titled '118031217797609 - ImportJob - Can't see status of the disk.' It includes a 'Support Request' button and a red box around a '+ New message' button.

## Next steps

Learn how to [Troubleshoot issues related to Import/Export service](#).

# End-to-end troubleshooting using Azure Storage metrics and logging, AzCopy, and Message Analyzer

1/21/2020 • 23 minutes to read • [Edit Online](#)

Diagnosing and troubleshooting is a key skill for building and supporting client applications with Microsoft Azure Storage. Due to the distributed nature of an Azure application, diagnosing and troubleshooting errors and performance issues may be more complex than in traditional environments.

In this tutorial, we demonstrate how to identify certain errors that may affect performance, and troubleshoot those errors from end-to-end using tools provided by Microsoft and Azure Storage, in order to optimize the client application.

This tutorial provides a hands-on exploration of an end-to-end troubleshooting scenario. For an in-depth conceptual guide to troubleshooting Azure storage applications, see [Monitor, diagnose, and troubleshoot Microsoft Azure Storage](#).

## Tools for troubleshooting Azure Storage applications

To troubleshoot client applications using Microsoft Azure Storage, you can use a combination of tools to determine when an issue has occurred and what the cause of the problem may be. These tools include:

- **Azure Storage Analytics.** [Azure Storage Analytics](#) provides metrics and logging for Azure Storage.
  - **Storage metrics** tracks transaction metrics and capacity metrics for your storage account. Using metrics, you can determine how your application is performing according to a variety of different measures. See [Storage Analytics Metrics Table Schema](#) for more information about the types of metrics tracked by Storage Analytics.
  - **Storage logging** logs each request to the Azure Storage services to a server-side log. The log tracks detailed data for each request, including the operation performed, the status of the operation, and latency information. See [Storage Analytics Log Format](#) for more information about the request and response data that is written to the logs by Storage Analytics.
- **Azure portal.** You can configure metrics and logging for your storage account in the [Azure portal](#). You can also view charts and graphs that show how your application is performing over time, and configure alerts to notify you if your application performs differently than expected for a specified metric.

See [Monitor a storage account in the Azure portal](#) for information about configuring monitoring in the Azure portal.

- **AzCopy.** Server logs for Azure Storage are stored as blobs, so you can use AzCopy to copy the log blobs to a local directory for analysis using Microsoft Message Analyzer. See [Transfer data with the AzCopy Command-Line Utility](#) for more information about AzCopy.
- **Microsoft Message Analyzer.** Message Analyzer is a tool that consumes log files and displays log data in a visual format that makes it easy to filter, search, and group log data into useful sets that you can use to analyze errors and performance issues. See [Microsoft Message Analyzer Operating Guide](#) for more information about Message Analyzer.

## About the sample scenario

For this tutorial, we'll examine a scenario where Azure Storage metrics indicates a low percent success rate for an application that calls Azure storage. The low percent success rate metric (shown as **PercentSuccess** in the [Azure](#)

[portal](#) and in the metrics tables) tracks operations that succeed, but that return an HTTP status code that is greater than 299. In the server-side storage log files, these operations are recorded with a transaction status of **ClientOtherErrors**. For more details about the low percent success metric, see [Metrics show low PercentSuccess or analytics log entries have operations with transaction status of ClientOtherErrors](#).

Azure Storage operations may return HTTP status codes greater than 299 as part of their normal functionality. But these errors in some cases indicate that you may be able to optimize your client application for improved performance.

In this scenario, we'll consider a low percent success rate to be anything below 100%. You can choose a different metric level, however, according to your needs. We recommend that during testing of your application, you establish a baseline tolerance for your key performance metrics. For example, you might determine, based on testing, that your application should have a consistent percent success rate of 90%, or 85%. If your metrics data shows that the application is deviating from that number, then you can investigate what may be causing the increase.

For our sample scenario, once we've established that the percent success rate metric is below 100%, we will examine the logs to find the errors that correlate to the metrics, and use them to figure out what is causing the lower percent success rate. We'll look specifically at errors in the 400 range. Then we'll more closely investigate 404 (Not Found) errors.

### Some causes of 400-range errors

The examples below shows a sampling of some 400-range errors for requests against Azure Blob Storage, and their possible causes. Any of these errors, as well as errors in the 300 range and the 500 range, can contribute to a low percent success rate.

Note that the lists below are far from complete. See [Status and Error Codes](#) on MSDN for details about general Azure Storage errors and about errors specific to each of the storage services.

#### Status Code 404 (Not Found) Examples

Occurs when a read operation against a container or blob fails because the blob or container is not found.

- Occurs if a container or blob has been deleted by another client before this request.
- Occurs if you are using an API call that creates the container or blob after checking whether it exists. The `CreateIfNotExists` APIs make a HEAD call first to check for the existence of the container or blob; if it does not exist, a 404 error is returned, and then a second PUT call is made to write the container or blob.

#### Status Code 409 (Conflict) Examples

- Occurs if you use a Create API to create a new container or blob, without checking for existence first, and a container or blob with that name already exists.
- Occurs if a container is being deleted, and you attempt to create a new container with the same name before the deletion operation is complete.
- Occurs if you specify a lease on a container or blob, and there is already a lease present.

#### Status Code 412 (Precondition Failed) Examples

- Occurs when the condition specified by a conditional header has not been met.
- Occurs when the lease ID specified does not match the lease ID on the container or blob.

## Generate log files for analysis

In this tutorial, we'll use Message Analyzer to work with three different types of log files, although you could choose to work with any one of these:

- The **server log**, which is created when you enable Azure Storage logging. The server log contains data about each operation called against one of the Azure Storage services - blob, queue, table, and file. The server log indicates which operation was called and what status code was returned, as well as other details about the

request and response.

- The **.NET client log**, which is created when you enable client-side logging from within your .NET application. The client log includes detailed information about how the client prepares the request and receives and processes the response.
- The **HTTP network trace log**, which collects data on HTTP/HTTPS request and response data, including for operations against Azure Storage. In this tutorial, we'll generate the network trace via Message Analyzer.

## Configure server-side logging and metrics

First, we'll need to configure Azure Storage logging and metrics, so that we have data from the service side to analyze. You can configure logging and metrics in a variety of ways - via the [Azure portal](#), by using PowerShell, or programmatically. See [Enable metrics](#) and [Enable logging](#) for details about configuring logging and metrics.

## Configure .NET client-side logging

To configure client-side logging for a .NET application, enable .NET diagnostics in the application's configuration file (web.config or app.config). See [Client-side Logging with the .NET Storage Client Library](#) and [Client-side Logging with the Microsoft Azure Storage SDK for Java](#) on MSDN for details.

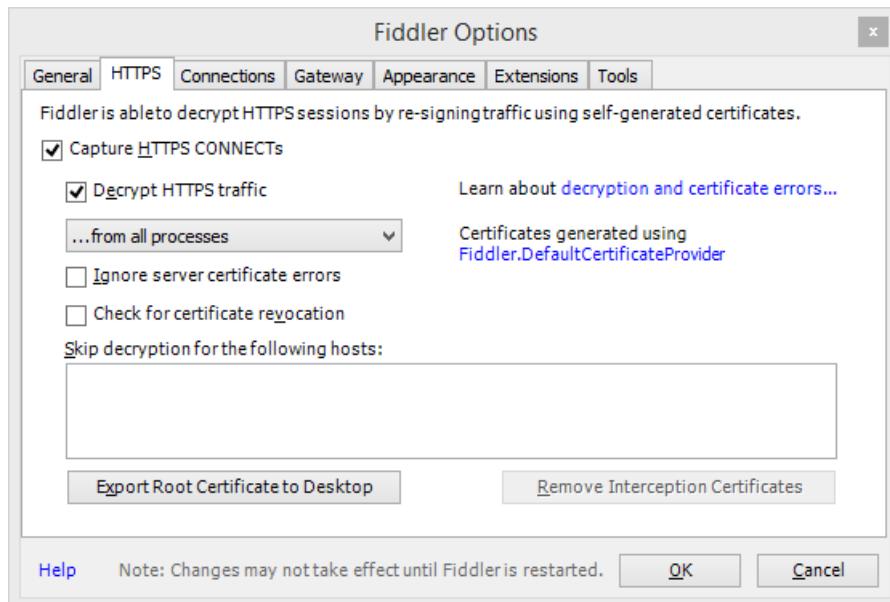
The client-side log includes detailed information about how the client prepares the request and receives and processes the response.

The Storage Client Library stores client-side log data in the location specified in the application's configuration file (web.config or app.config).

## Collect a network trace

You can use Message Analyzer to collect an HTTP/HTTPS network trace while your client application is running. Message Analyzer uses [Fiddler](#) on the back end. Before you collect the network trace, we recommend that you configure Fiddler to record unencrypted HTTPS traffic:

1. Install [Fiddler](#).
2. Launch Fiddler.
3. Select **Tools | Fiddler Options**.
4. In the Options dialog, ensure that **Capture HTTPS CONNECTs** and **Decrypt HTTPS Traffic** are both selected, as shown below.



For the tutorial, collect and save a network trace first in Message Analyzer, then create an analysis session to analyze the trace and the logs. To collect a network trace in Message Analyzer:

1. In Message Analyzer, select **File | Quick Trace | Unencrypted HTTPS**.

2. The trace will begin immediately. Select **Stop** to stop the trace so that we can configure it to trace storage traffic only.
3. Select **Edit** to edit the tracing session.
4. Select the **Configure** link to the right of the Microsoft-Pef-WebProxy ETW provider.
5. In the **Advanced Settings** dialog, click the **Provider** tab.
6. In the **Hostname Filter** field, specify your storage endpoints, separated by spaces. For example, you can specify your endpoints as follows; change `storagesample` to the name of your storage account:

```
storagesample.blob.core.windows.net storagesample.queue.core.windows.net
storagesample.table.core.windows.net
```
7. Exit the dialog, and click **Restart** to begin collecting the trace with the hostname filter in place, so that only Azure Storage network traffic is included in the trace.

**NOTE**

After you have finished collecting your network trace, we strongly recommend that you revert the settings that you may have changed in Fiddler to decrypt HTTPS traffic. In the Fiddler Options dialog, deselect the **Capture HTTPS CONNECTs** and **Decrypt HTTPS Traffic** checkboxes.

See [Using the Network Tracing Features](#) on Technet for more details.

## Review metrics data in the Azure portal

Once your application has been running for a period of time, you can review the metrics charts that appear in the [Azure portal](#) to observe how your service has been performing.

First, navigate to your storage account in the Azure portal. By default, a monitoring chart with the **Success percentage** metric is displayed on the account blade. If you've previously modified the chart to display different metrics, add the **Success percentage** metric.

You'll now see **Success percentage** in the monitoring chart, along with any other metrics you may have added. In the scenario we'll investigate next by analyzing the logs in Message Analyzer, the percent success rate is somewhat below 100%.

For more details on adding and customizing metrics charts, see [Customize metrics charts](#).

**NOTE**

It may take some time for your metrics data to appear in the Azure portal after you enable storage metrics. This is because hourly metrics for the previous hour are not displayed in the Azure portal until the current hour has elapsed. Also, minute metrics are not currently displayed in the Azure portal. So depending on when you enable metrics, it may take up to two hours to see metrics data.

## Use AzCopy to copy server logs to a local directory

Azure Storage writes server log data to blobs, while metrics are written to tables. Log blobs are available in the well-known `$logs` container for your storage account. Log blobs are named hierarchically by year, month, day, and hour, so that you can easily locate the range of time you wish to investigate. For example, in the `storagesample` account, the container for the log blobs for 01/02/2015, from 8-9 am, is

[https://storagesample.blob.core.windows.net/\\$logs/blob/2015/01/08/0800](https://storagesample.blob.core.windows.net/$logs/blob/2015/01/08/0800). The individual blobs in this container are named sequentially, beginning with `000000.log`.

You can use the AzCopy command-line tool to download these server-side log files to a location of your choice on your local machine. For example, you can use the following command to download the log files for blob operations that took place on January 2, 2015 to the folder `C:\Temp\Logs\Server`; replace `<storageaccountname>` with the name of your storage account:

```
azcopy copy 'http://<storageaccountname>.blob.core.windows.net/$logs/blob/2015/01/02' 'C:\Temp\Logs\Server' --recursive
```

AzCopy is available for download on the [Azure Downloads](#) page. For details about using AzCopy, see [Transfer data with the AzCopy Command-Line Utility](#).

For additional information about downloading server-side logs, see [Download Storage Logging log data](#).

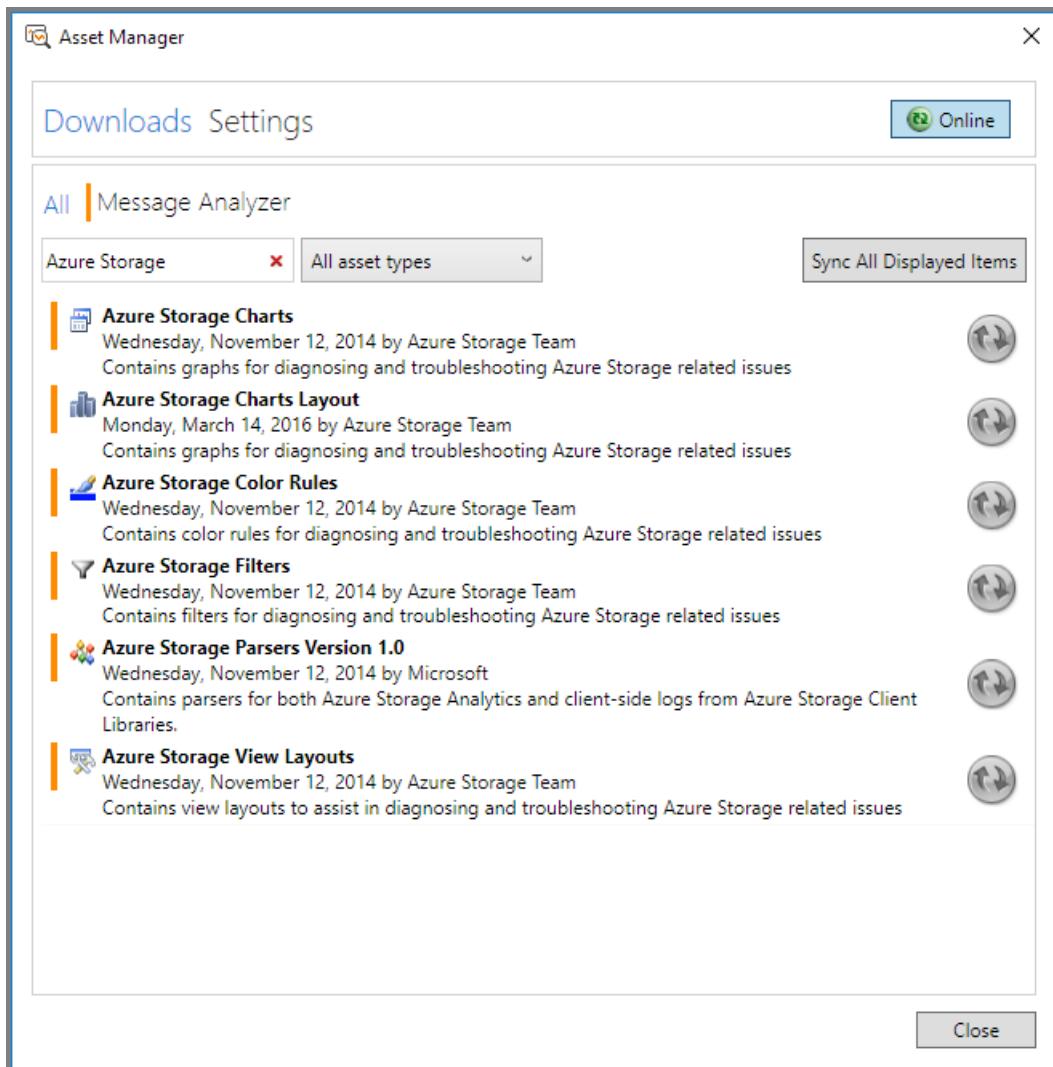
## Use Microsoft Message Analyzer to analyze log data

Microsoft Message Analyzer is a tool for capturing, displaying, and analyzing protocol messaging traffic, events, and other system or application messages in troubleshooting and diagnostic scenarios. Message Analyzer also enables you to load, aggregate, and analyze data from log and saved trace files. For more information about Message Analyzer, see [Microsoft Message Analyzer Operating Guide](#).

Message Analyzer includes assets for Azure Storage that help you to analyze server, client, and network logs. In this section, we'll discuss how to use those tools to address the issue of low percent success in the storage logs.

### Download and install Message Analyzer and the Azure Storage Assets

1. Download [Message Analyzer](#) from the Microsoft Download Center, and run the installer.
2. Launch Message Analyzer.
3. From the Tools menu, select **Asset Manager**. In the **Asset Manager** dialog, select **Downloads**, then filter on **Azure Storage**. You will see the Azure Storage Assets, as shown in the picture below.
4. Click **Sync All Displayed Items** to install the Azure Storage Assets. The available assets include:
  - **Azure Storage Color Rules:** Azure Storage color rules enable you to define special filters that use color, text, and font styles to highlight messages that contain specific information in a trace.
  - **Azure Storage Charts:** Azure Storage charts are predefined charts that graph server log data. Note that to use Azure Storage charts at this time, you may only load the server log into the Analysis Grid.
  - **Azure Storage Parsers:** The Azure Storage parsers parse the Azure Storage client, server, and HTTP logs in order to display them in the Analysis Grid.
  - **Azure Storage Filters:** Azure Storage filters are predefined criteria that you can use to query your data in the Analysis Grid.
  - **Azure Storage View Layouts:** Azure Storage view layouts are predefined column layouts and groupings in the Analysis Grid.
5. Restart Message Analyzer after you've installed the assets.



#### NOTE

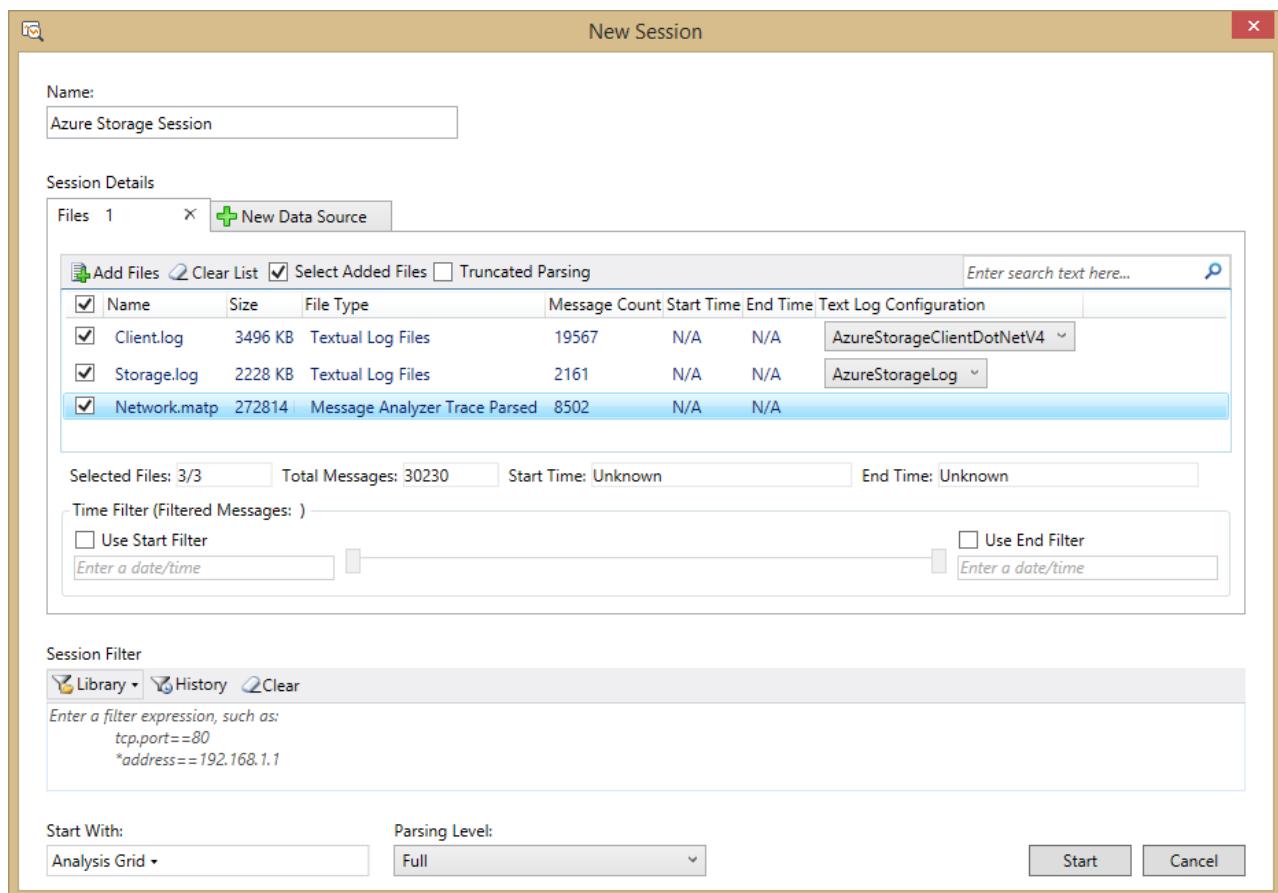
Install all of the Azure Storage assets shown for the purposes of this tutorial.

### Import your log files into Message Analyzer

You can import all of your saved log files (server-side, client-side, and network) into a single session in Microsoft Message Analyzer for analysis.

1. On the **File** menu in Microsoft Message Analyzer, click **New Session**, and then click **Blank Session**. In the **New Session** dialog, enter a name for your analysis session. In the **Session Details** panel, click on the **Files** button.
2. To load the network trace data generated by Message Analyzer, click on **Add Files**, browse to the location where you saved your .matp file from your web tracing session, select the .matp file, and click **Open**.
3. To load the server-side log data, click on **Add Files**, browse to the location where you downloaded your server-side logs, select the log files for the time range you want to analyze, and click **Open**. Then, in the **Session Details** panel, set the **Text Log Configuration** drop-down for each server-side log file to **AzureStorageLog** to ensure that Microsoft Message Analyzer can parse the log file correctly.
4. To load the client-side log data, click on **Add Files**, browse to the location where you saved your client-side logs, select the log files you want to analyze, and click **Open**. Then, in the **Session Details** panel, set the **Text Log Configuration** drop-down for each client-side log file to **AzureStorageClientDotNetV4** to ensure that Microsoft Message Analyzer can parse the log file correctly.
5. Click **Start** in the **New Session** dialog to load and parse the log data. The log data displays in the Message Analyzer Analysis Grid.

The picture below shows an example session configured with server, client, and network trace log files.



Note that Message Analyzer loads log files into memory. If you have a large set of log data, you will want to filter it in order to get the best performance from Message Analyzer.

First, determine the time frame that you are interested in reviewing, and keep this time frame as small as possible. In many cases you will want to review a period of minutes or hours at most. Import the smallest set of logs that can meet your needs.

If you still have a large amount of log data, then you may want to specify a session filter to filter your log data before you load it. In the **Session Filter** box, select the **Library** button to choose a predefined filter; for example, choose **Global Time Filter I** from the Azure Storage filters to filter on a time interval. You can then edit the filter criteria to specify the starting and ending timestamp for the interval you want to see. You can also filter on a particular status code; for example, you can choose to load only log entries where the status code is 404.

For more information about importing log data into Microsoft Message Analyzer, see [Retrieving Message Data](#) on TechNet.

### Use the client request ID to correlate log file data

The Azure Storage Client Library automatically generates a unique client request ID for every request. This value is written to the client log, the server log, and the network trace, so you can use it to correlate data across all three logs within Message Analyzer. See [Client request ID](#) for additional information about the client request ID.

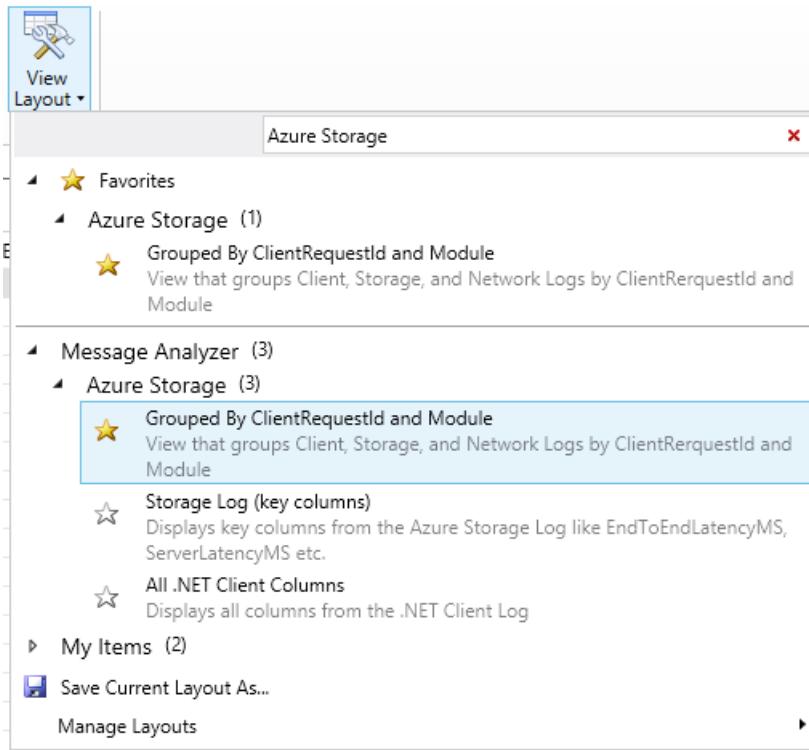
The sections below describe how to use pre-configured and custom layout views to correlate and group data based on the client request ID.

### Select a view layout to display in the Analysis Grid

The Storage Assets for Message Analyzer include Azure Storage View Layouts, which are pre-configured views that you can use to display your data with useful groupings and columns for different scenarios. You can also create custom view layouts and save them for reuse.

The picture below shows the **View Layout** menu, available by selecting **View Layout** from the toolbar ribbon. The

view layouts for Azure Storage are grouped under the **Azure Storage** node in the menu. You can search for **Azure Storage** in the search box to filter on Azure Storage view layouts only. You can also select the star next to a view layout to make it a favorite and display it at the top of the menu.



To begin with, select **Grouped by ClientRequestId and Module**. This view layout groups log data from all three logs first by client request ID, then by source log file (or **Module** in Message Analyzer). With this view, you can drill down into a particular client request ID, and see data from all three log files for that client request ID.

The picture below shows this layout view applied to the sample log data, with a subset of columns displayed. You can see that for a particular client request ID, the Analysis Grid displays data from the client log, server log, and network trace.

ClientRequestId	Module				
MessageNumber	Timestamp	TimeElapse	Source	Destination	Module
<b>ClientRequestId (3): 00d98e5a-cc8f-4a4b-8dc1-5a635d9d7265</b>					
<b>Module (9): AzureStorageClientDotNetV4</b>					
17271					AzureStorageClientDotNetV4
17272					AzureStorageClientDotNetV4
17273					AzureStorageClientDotNetV4
17274					AzureStorageClientDotNetV4
17275					AzureStorageClientDotNetV4
17276					AzureStorageClientDotNetV4
17277					AzureStorageClientDotNetV4
17278					AzureStorageClientDotNetV4
17279					AzureStorageClientDotNetV4
<b>Module (1): AzureStorageLog</b>					
1732	2014-10-20T16:39:43.3627051				AzureStorageLog
<b>Module (1): HTTP</b>					
7791	2014-10-20T16:39:42.5689042	0.0319180	Local	photouploadercs.blob.cor...	HTTP
<b>ClientRequestId (3): 0103dde1-8391-468f-ae87-f95d30718b2b</b>					
<b>ClientRequestId (3): 011652df-76b2-4ecd-8e92-819ac91087e6</b>					

#### NOTE

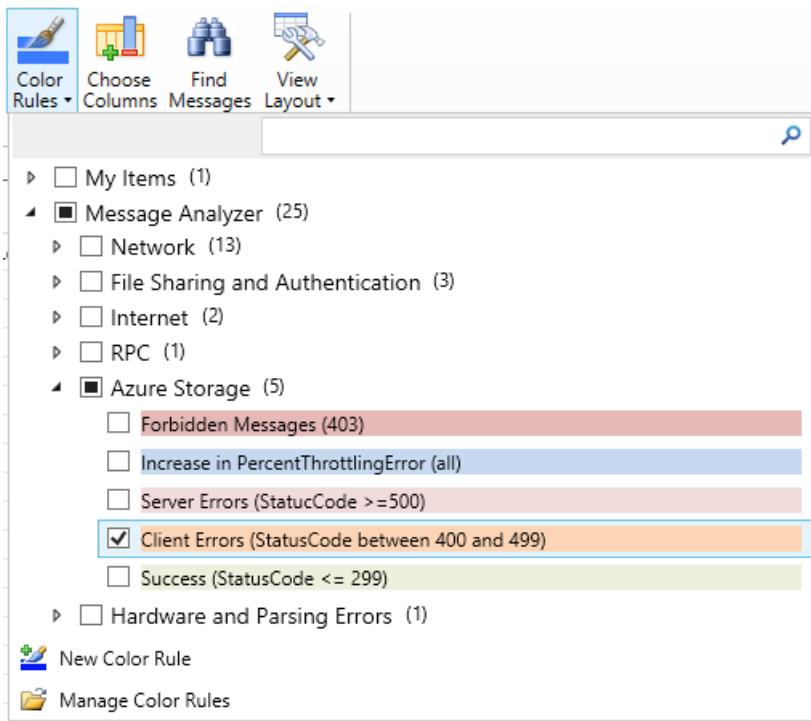
Different log files have different columns, so when data from multiple log files is displayed in the Analysis Grid, some columns may not contain any data for a given row. For example, in the picture above, the client log rows do not show any data for the **Timestamp**, **TimeElapsed**, **Source**, and **Destination** columns, because these columns do not exist in the client log, but do exist in the network trace. Similarly, the **Timestamp** column displays timestamp data from the server log, but no data is displayed for the **TimeElapsed**, **Source**, and **Destination** columns, which are not part of the server log.

In addition to using the Azure Storage view layouts, you can also define and save your own view layouts. You can select other desired fields for grouping data and save the grouping as part of your custom layout as well.

#### Apply color rules to the Analysis Grid

The Storage Assets also include color rules, which offer a visual means to identify different types of errors in the Analysis Grid. The predefined color rules apply to HTTP errors, so they appear only for the server log and network trace.

To apply color rules, select **Color Rules** from the toolbar ribbon. You'll see the Azure Storage color rules in the menu. For the tutorial, select **Client Errors (StatusCode between 400 and 499)**, as shown in the picture below.



In addition to using the Azure Storage color rules, you can also define and save your own color rules.

#### Group and filter log data to find 400-range errors

Next, we'll group and filter the log data to find all errors in the 400 range.

1. Locate the **StatusCode** column in the Analysis Grid, right-click the column heading, and select **Group**.
2. Next, group on the **ClientRequestId** column. You'll see that the data in the Analysis Grid is now organized by status code and by client request ID.
3. Display the View Filter tool window if it is not already displayed. On the toolbar ribbon, select **Tool Windows**, then **View Filter**.
4. To filter the log data to display only 400-range errors, add the following filter criteria to the **View Filter** window, and click **Apply**:

```
(AzureStorageLog.StatusCode >= 400 && AzureStorageLog.StatusCode <=499) || (HTTP.StatusCode >= 400 && HTTP.StatusCode <= 499)
```

The picture below shows the results of this grouping and filter. Expanding the ClientRequestId field beneath the grouping for status code 409, for example, shows an operation that resulted in that status code.

MessageNumber	Timestamp	Summary
StatusCode (77): 404		
StatusCode (6): 409		
ClientRequestId (2): 0c940cc5-a7b6-4f7c-87af-5e452c6cfa90		
4575 2014-10-20T16:38:53.9350134 Operation, Status: The specified blob already exists. (409), PUT http://photouploadercs.blob.c...		
1181 2014-10-20T16:38:54.0141050 PutBlob		
ClientRequestId (2): 443aa117-6b6d-4c7c-8b1d-0919449378a1		
ClientRequestId (2): acc91f3e-0ee2-431a-8d4d-618dca19020c		
ClientRequestId (2): bfbd9d10-f97a-4b97-9728-7a2f4df03656		
ClientRequestId (2): c23d2ddf-38ca-4d71-9a07-786bc74b8d06		
ClientRequestId (2): f00f7507-b2b8-4771-9ecd-075513ba8fdb		
StatusCode (37): 412		
ClientRequestId (2): 0588814e-38f2-4c58-9087-2717285004d5		
4557 2014-10-20T16:38:53.6224286 Operation, Status: The condition specified using HTTP conditional header(s) is not met. (412),...		
1172 2014-10-20T16:38:53.7010738 PutBlob		
ClientRequestId (2): 12869d87-cde1-464a-9324-2e5e0d1b68b7		

After applying this filter, you'll see that rows from the client log are excluded, as the client log does not include a **StatusCode** column. To begin with, we'll review the server and network trace logs to locate 404 errors, and then we'll return to the client log to examine the client operations that led to them.

#### NOTE

You can filter on the **StatusCode** column and still display data from all three logs, including the client log, if you add an expression to the filter that includes log entries where the status code is null. To construct this filter expression, use:

```
*StatusCode >= 400 or !*StatusCode
```

This filter returns all rows from the client log and only rows from the server log and HTTP log where the status code is greater than 400. If you apply it to the view layout grouped by client request ID and module, you can search or scroll through the log entries to find ones where all three logs are represented.

#### Filter log data to find 404 errors

The Storage Assets include predefined filters that you can use to narrow log data to find the errors or trends you are looking for. Next, we'll apply two predefined filters: one that filters the server and network trace logs for 404 errors, and one that filters the data on a specified time range.

1. Display the View Filter tool window if it is not already displayed. On the toolbar ribbon, select **Tool Windows**, then **View Filter**.
2. In the View Filter window, select **Library**, and search on **Azure Storage** to find the Azure Storage filters. Select the filter for **404 (Not Found) messages in all logs**.
3. Display the **Library** menu again, and locate and select the **Global Time Filter**.
4. Edit the timestamps shown in the filter to the range you wish to view. This will help to narrow the range of data to analyze.
5. Your filter should appear similar to the example below. Click **Apply** to apply the filter to the Analysis Grid.

```
((AzureStorageLog.StatusCode == 404 || HTTP.StatusCode == 404)) And (#Timestamp >= 2014-10-20T16:36:38
and #Timestamp <= 2014-10-20T16:36:39)
```

ClientRequestId			
MessageNumber	Timestamp	Summary	
ClientRequestId (2): 0b3d195b-aa64-42f3-ad58-c69632080f36	2014-10-20T16:36:38.8480664	Operation, Status: The specified blob does not exist. (404), GET http://	
3193	2014-10-20T16:36:38.9245975	GetBlob	
ClientRequestId (2): 58ca84bb-2953-4946-83af-e7ee014038cf	2014-10-20T16:36:38.0440455	Operation, Status: The specified blob does not exist. (404), GET http://	
3151	2014-10-20T16:36:38.1205171	GetBlob	
ClientRequestId (2): 0dbe2457-2a04-4676-914c-19a52f3cab62			
ClientRequestId (1): 2de8f2e5-5b7c-4092-9b5b-6bb12522bb13			
ClientRequestId (2): 6c9cf7ae-f85a-433f-968f-d32d31ce0538			
ClientRequestId (2): 73dc48c9-d209-48cd-9ed1-6b8ed6d58c2c			
ClientRequestId (2): dcba489ab-cdcf-407a-9fd8-22e02a506319			
ClientRequestId (2): f666f045-7f2f-4913-a770-e148840ff408			

## Analyze your log data

Now that you have grouped and filtered your data, you can examine the details of individual requests that generated 404 errors. In the current view layout, the data is grouped by client request ID, then by log source. Since we are filtering on requests where the StatusCode field contains 404, we'll see only the server and network trace data, not the client log data.

The picture below shows a specific request where a Get Blob operation yielded a 404 because the blob did not exist. Note that some columns have been removed from the standard view in order to display the relevant data.

ClientRequestId				Module
MessageNumber	Timestamp	Module	Summary	StatusCode
ClientRequestId (2): 2db35a12-b9f8-42d8-853e-8b60e18b22d				
411	2014-10-20T16:36:36.1403190	AzureStorageLog	GetBlob	404
ClientRequestId (2): 2de8f2e5-5b7c-4092-9b5b-6bb12522bb13				
3045	2014-10-20T16:36:36.0630581	HTTP	Operation, Status: The specified blob does not exist. (404), GET http... 404	
ClientRequestId (2): 3a89917e-24ff-4ed9-bd60-991670442cca				
ClientRequestId (2): 3e30de3b-abe1-4237-928d-1384ceb02e26				

Next, we'll correlate this client request ID with the client log data to see what actions the client was taking when the error happened. You can display a new Analysis Grid view for this session to view the client log data, which opens in a second tab:

- First, copy the value of the **ClientRequestId** field to the clipboard. You can do this by selecting either row, locating the **ClientRequestId** field, right-clicking on the data value, and choosing **Copy 'ClientRequestId'**.
- On the toolbar ribbon, select **New Viewer**, then select **Analysis Grid** to open a new tab. The new tab shows all data in your log files, without grouping, filtering, or color rules.
- On the toolbar ribbon, select **View Layout**, then select **All .NET Client Columns** under the **Azure Storage** section. This view layout shows data from the client log as well as the server and network trace logs. By default it is sorted on the **MessageNumber** column.
- Next, search the client log for the client request ID. On the toolbar ribbon, select **Find Messages**, then specify a custom filter on the client request ID in the **Find** field. Use this syntax for the filter, specifying your own client request ID:

```
*ClientRequestId == "398bac41-7725-484b-8a69-2a9e48fc669a"
```

Message Analyzer locates and selects the first log entry where the search criteria matches the client request ID. In the client log, there are several entries for each client request ID, so you may want to group them on the **ClientRequestId** field to make it easier to see them all together. The picture below shows all of the messages in the client log for the specified client request ID.

Start Page	Azure Storage Se...	Azure Storage Se... X	
*ClientRequestId == "398bac41-7725-484b-8a69-2a9e48fc669a"			Find Previous Library ▾ Find History ▾
<b>ClientRequestId</b>			
	MessageNumber	ClientRequestId	Level LevelString Description
ClientRequestId (11): 39616321-08f2-4ce0-b1a8-45ac1fff182b4			
ClientRequestId (13): 3986bd72-5a8c-4f0d-8730-d5bf1536112			
ClientRequestId (12): 398bac41-7725-484b-8a69-2a9e48fc669a			
3803	398bac41-7725-484b-8a69-2a9e48fc669a	3	Information Starting operation with location Primary per location mode PrimaryOnly.
3804	398bac41-7725-484b-8a69-2a9e48fc669a	3	Information Starting synchronous request to http://photouploadercs.blob.core.windows.net/testclde9e5dad9c54fc6b0...
3805	398bac41-7725-484b-8a69-2a9e48fc669a	4	Verbose StringToSign = GET.....x-ms-client-request-id:398bac41-7725-484b-8a69-2a9e48fc669a.x-ms-date:...
3806	398bac41-7725-484b-8a69-2a9e48fc669a	3	Information Waiting for response.
3807	398bac41-7725-484b-8a69-2a9e48fc669a	2	Warning Exception thrown while waiting for response: The remote server returned an error: (404) Not Found..
3808	398bac41-7725-484b-8a69-2a9e48fc669a	3	Information Response received. Status code = 404, Request ID = 793143c6-0001-0029-1c50-74ec56000000, Content-MD5...
3809	398bac41-7725-484b-8a69-2a9e48fc669a	2	Warning Exception thrown during the operation: The remote server returned an error: (404) Not Found..
3810	398bac41-7725-484b-8a69-2a9e48fc669a	3	Information Checking if the operation should be retried. Retry count = 0, HTTP status code = 404, Retryable exce...
3811	398bac41-7725-484b-8a69-2a9e48fc669a	3	Information The next location has been set to Primary, based on the location mode.
3812	398bac41-7725-484b-8a69-2a9e48fc669a	1	Error Retry policy did not allow for a retry. Failing With The remote server returned an error: (404) Not...
3805	398bac41-7725-484b-8a69-2a9e48fc669a		
411	398bac41-7725-484b-8a69-2a9e48fc669a		
ClientRequestId (13): 399fec04-555e-4b81-900d-f178b4a8f352			
ClientRequestId (9): 39b83db1-8409-4b37-be31-bf38a3fc86a			
ClientRequestId (11): 39cf9f20-a6d4-43b2-b67d-9785c17e031			

Using the data shown in the view layouts in these two tabs, you can analyze the request data to determine what may have caused the error. You can also look at requests that preceded this one to see if a previous event may have led to the 404 error. For example, you can review the client log entries preceding this client request ID to determine whether the blob may have been deleted, or if the error was due to the client application calling a `CreateIfNotExists` API on a container or blob. In the client log, you can find the blob's address in the **Description** field; in the server and network trace logs, this information appears in the **Summary** field.

Once you know the address of the blob that yielded the 404 error, you can investigate further. If you search the log entries for other messages associated with operations on the same blob, you can check whether the client previously deleted the entity.

## Analyze other types of storage errors

Now that you are familiar with using Message Analyzer to analyze your log data, you can analyze other types of errors using view layouts, color rules, and searching/filtering. The tables below lists some issues you may encounter and the filter criteria you can use to locate them. For more information on constructing filters and the Message Analyzer filtering language, see [Filtering Message Data](#).

TO INVESTIGATE...	USE FILTER EXPRESSION...	EXPRESSION APPLIES TO LOG (CLIENT, SERVER, NETWORK, ALL)
Unexpected delays in message delivery on a queue	AzureStorageClientDotNetV4.Description n contains "Retrying failed operation."	Client
HTTP Increase in PercentThrottlingError	HttpResponse.StatusCode == 500    HttpResponse.StatusCode == 503	Network
Increase in PercentTimeoutError	HttpResponse.StatusCode == 500	Network
Increase in PercentTimeoutError (all)	*StatusCode == 500	All
Increase in PercentNetworkError	AzureStorageClientDotNetV4.EventLogEntry.Level < 2	Client
HTTP 403 (Forbidden) messages	HttpResponse.StatusCode == 403	Network
HTTP 404 (Not found) messages	HttpResponse.StatusCode == 404	Network
404 (all)	*StatusCode == 404	All

TO INVESTIGATE...	USE FILTER EXPRESSION...	EXPRESSION APPLIES TO LOG (CLIENT, SERVER, NETWORK, ALL)
Shared Access Signature (SAS) authorization issue	AzureStorageLog.RequestStatus == "SASAuthorizationError"	Network
HTTP 409 (Conflict) messages	HTTPResponse.StatusCode == 409	Network
409 (all)	*StatusCode == 409	All
Low PercentSuccess or analytics log entries have operations with transaction status of ClientOtherErrors	AzureStorageLog.RequestStatus == "ClientOtherError"	Server
Nagle Warning	((AzureStorageLog.EndToEndLatencyMS - AzureStorageLog.ServerLatencyMS) > (AzureStorageLog.ServerLatencyMS * 1.5)) and (AzureStorageLog.RequestPacketSize < 1460) and (AzureStorageLog.EndToEndLatencyMS - AzureStorageLog.ServerLatencyMS) >= 200)	Server
Range of time in Server and Network logs	#Timestamp >= 2014-10-20T16:36:38 and #Timestamp <= 2014-10-20T16:36:39	Server, Network
Range of time in Server logs	AzureStorageLog.Timestamp >= 2014-10-20T16:36:38 and AzureStorageLog.Timestamp <= 2014-10-20T16:36:39	Server

## Next steps

For more information about troubleshooting end-to-end scenarios in Azure Storage, see these resources:

- [Monitor, diagnose, and troubleshoot Microsoft Azure Storage](#)
- [Storage Analytics](#)
- [Monitor a storage account in the Azure portal](#)
- [Transfer data with the AzCopy Command-Line Utility](#)
- [Microsoft Message Analyzer Operating Guide](#)

# Monitor, diagnose, and troubleshoot Microsoft Azure Storage

4/17/2020 • 55 minutes to read • [Edit Online](#)

## 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.

### NOTE

Azure Files does not support logging at this time.

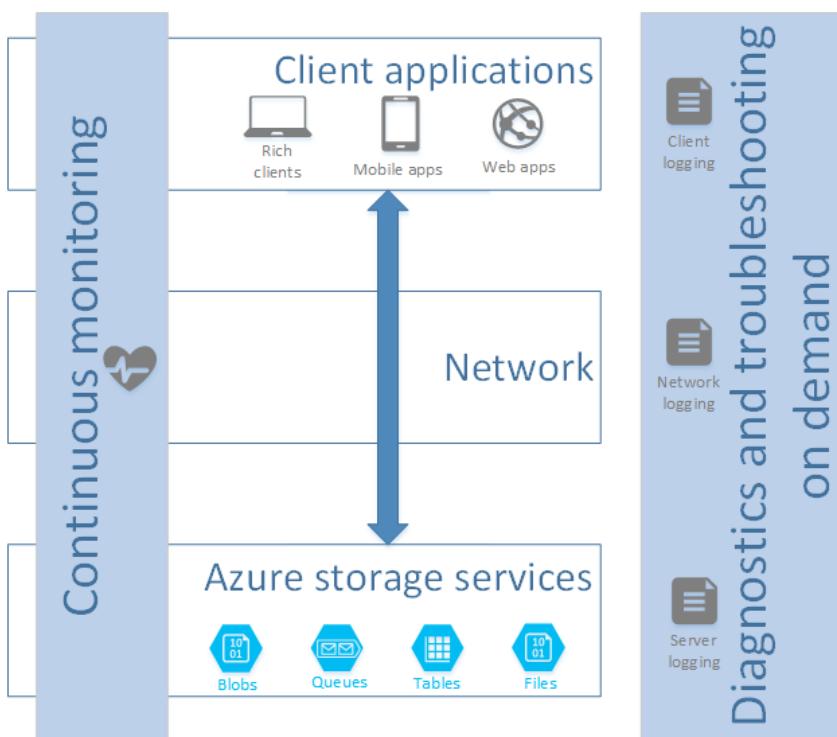
For a hands-on guide to end-to-end troubleshooting in Azure Storage applications, see [End-to-End Troubleshooting using Azure Storage Metrics and Logging, AzCopy, and Message Analyzer](#).

- [Introduction](#)
  - [How this guide is organized](#)
- [Monitoring your storage service](#)
  - [Monitoring service health](#)
  - [Monitoring capacity](#)
  - [Monitoring availability](#)
  - [Monitoring performance](#)
- [Diagnosing storage issues](#)
  - [Service health issues](#)
  - [Performance issues](#)
  - [Diagnosing errors](#)
  - [Storage emulator issues](#)
  - [Storage logging tools](#)
  - [Using network logging tools](#)
- [End-to-end tracing](#)
  - [Correlating log data](#)
  - [Client request ID](#)
  - [Server request ID](#)
  - [Timestamps](#)
- [Troubleshooting guidance](#)
  - [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](#)
  - [Metrics show an increase in PercentThrottlingError](#)
  - [Metrics show an increase in PercentTimeoutError](#)
  - [Metrics show an increase in PercentNetworkError](#)
  - [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
- Troubleshooting VHDs on Windows virtual machines
- Troubleshooting VHDs on Linux virtual machines
- Troubleshooting Azure Files issues with Windows
- Troubleshooting Azure Files issues with Linux
- Appendices
  - Appendix 1: Using Fiddler to capture HTTP and HTTPS traffic
  - Appendix 2: Using Wireshark to capture network traffic
  - Appendix 3: Using Microsoft Message Analyzer to capture network traffic
  - Appendix 4: Using Excel to view metrics and log data
  - Appendix 5: Monitoring with Application Insights for Azure DevOps

## Introduction

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.

### 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, Fiddler for analyzing HTTP/HTTPS messages, and Microsoft Message Analyzer for correlating log data.

## 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.

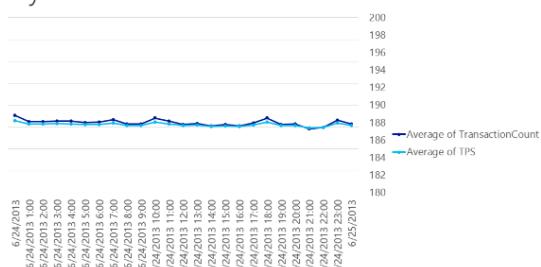
In the Azure portal, you can view metrics such as availability, total requests, and average latency numbers for a storage account. A notification rule has also been set up to alert an administrator if availability drops below a certain level. From viewing this data, one possible area for investigation 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:

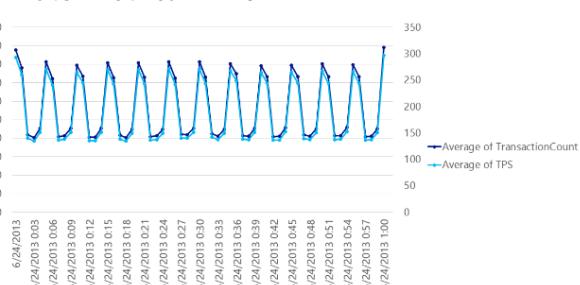
- Establishing some baseline metrics for application that will enable you to compare current data and identify any significant changes in the behavior of Azure storage and your application. The values of your baseline metrics will, in many cases, be application specific and you should establish them 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.

Hourly metrics - TPS



Minute Metrics - TPS



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](#).

While the [Azure portal](#) collects health information from inside the Azure datacenters (inside-out monitoring), you could also consider adopting an outside-in approach to generate synthetic transactions that periodically access your Azure-hosted web application from

multiple locations. The services offered by [Dynatrace](#) and Application Insights for Azure DevOps are examples of this approach. 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"](#).
- Microsoft Message Analyzer is a tool from Microsoft that supersedes Netmon and that in addition to capturing network packet data, helps you to view and analyze the log data captured from other tools. For more information, see ["Appendix 3: Using Microsoft Message Analyzer 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. Message Analyzer can operate at both HTTP and TCP layers.

## 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.

If the Storage Client Library throws a **StorageException** in the client, the **RequestInformation** property contains a **RequestResult** object that includes a **ServiceRequestId** property. You can also access a **RequestResult** object from an **OperationContext** instance.

The code sample below demonstrates how to set a custom **ClientRequestId** value by attaching an **OperationContext** object the request to the storage service. It also shows how to retrieve the **ServerRequestId** value from the response message.

```
//Parse the connection string for the storage account.
const string ConnectionString = "DefaultEndpointsProtocol=https;AccountName=account-name;AccountKey=account-key";
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(ConnectionString);
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

// Create an Operation Context that includes custom ClientRequestId string based on constants defined within the application along
with a Guid.
OperationContext oc = new OperationContext();
oc.ClientRequestId = String.Format("{0} {1} {2} {3}", HOSTNAME, APPNAME, USERID, Guid.NewGuid().ToString());

try
{
 CloudBlobContainer container = blobClient.GetContainerReference("democontainer");
 ICloudBlob blob = container.GetBlobReferenceFromServer("testImage.jpg", null, null, oc);
 var downloadToPath = string.Format("./{0}", blob.Name);
 using (var fs = File.OpenWrite(downloadToPath))
 {
 blob.DownloadToStream(fs, null, null, oc);
 Console.WriteLine("\t Blob downloaded to file: {0}", downloadToPath);
 }
}
catch (StorageException storageException)
{
 Console.WriteLine("Storage exception {0} occurred", storageException.Message);
 // Multiple results may exist due to client side retry logic - each retried operation will have a unique ServiceRequestId
 foreach (var result in oc.RequestResults)
 {
 Console.WriteLine("HttpStatus: {0}, ServiceRequestId {1}", result.HttpStatusCode, result.ServiceRequestId);
 }
}
```

#### 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

[You are experiencing unexpected reboots of Virtual Machines that have a large number of attached VHDs]

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**.

METRIC	AVERAGE	MINIMUM	MAXIMUM	TOTAL
AverageE2ELatency	49.47 ms	0 ms	139.65 ms	--
AverageServerLatency	2.49 ms	0 ms	4.59 ms	--

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 acknowledgement 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.

```
var storageAccount = CloudStorageAccount.Parse(connStr);
ServicePoint tableServicePoint = ServicePointManager.FindServicePoint(storageAccount.TableEndpoint);
tableServicePoint.UseNagleAlgorithm = false;
ServicePoint queueServicePoint = ServicePointManager.FindServicePoint(storageAccount.QueueEndpoint);
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 or Microsoft Message Analyzer.

For more information about using Wireshark to troubleshoot network issues, see "[Appendix 2: Using Wireshark to capture network traffic](#)."

For more information about using Microsoft Message Analyzer to troubleshoot network issues, see "[Appendix 3: Using Microsoft Message Analyzer 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 or Microsoft Message Analyzer to investigate network issues.

For more information about using Wireshark to troubleshoot network issues, see "[Appendix 2: Using Wireshark to capture network traffic](#)."

For more information about using Microsoft Message Analyzer to troubleshoot network issues, see "[Appendix 3: Using Microsoft Message Analyzer 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 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 acknowledgement, 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 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, Microsoft Message Analyzer, 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://domemaildist.blob.core.windows.net/azureimblblobcontainer/blobCreatedViaSAS.txt?sv=2014-02-14&amp;sr=c&amp;si=mypolicy&amp;sig=OFnd4Rd7z01flvh%2BmcR6zbudIH2F5Ikrm%2FyhNYZEmJNQ%3D&amp;api-version=2014-02-14">https://domemaildist.blob.core.windows.net/azureimblblobcontainer/blobCreatedViaSAS.txt?sv=2014-02-14&amp;sr=c&amp;si=mypolicy&amp;sig=OFnd4Rd7z01flvh%2BmcR6zbudIH2F5Ikrm%2FyhNYZEmJNQ%3D&amp;api-version=2014-02-14</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 = 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..

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 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-...	<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.

REQUEST ID	OPERATION
e2d06d78...	CreateIfNotExists method to create the blob container. Note that this operation includes a HEAD request that checks for the existence of the container. The HEAD returns a 404 message but continues.
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 <a href="https://domemaildist.blob.core.windows.net/azuremmblobcontainer">https://domemaildist.blob.core.windows.net/azuremmblobcontainer</a> .
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 <a href="https://domemaildist.blob.core.windows.net/azuremmblobcontainer">https://domemaildist.blob.core.windows.net/azuremmblobcontainer</a> .
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 = .
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 <a href="https://domemaildist.blob.core.windows.net/azuremmblobcontainer">https://domemaildist.blob.core.windows.net/azuremmblobcontainer</a> .
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 <a href="https://domemaildist.blob.core.windows.net/azuremmblobcontainer/blobCreate">https://domemaildist.blob.core.windows.net/azuremmblobcontainer/blobCreate</a> .

REQUEST ID	OPERATION TEXT
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 <a href="https://domemaildist.blob.core.windows.net/azuremmblobcontainer">https://domemaildist.blob.core.windows.net/azuremmblobcontainer</a> .
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 = .
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, 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?sv=2014-02-14&amp;sr=c&amp;si=mypolicy&amp;sig=XXXXX&amp;api-version=2014-02-14">https://domemaildist.blob.core.windows.net/azureimblobcontainer/blobCreate?sv=2014-02-14&amp;sr=c&amp;si=mypolicy&amp;sig=XXXXX&amp;api-version=2014-02-14</a>
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:

```
CloudBlobClient client = new CloudBlobClient(blobEndpoint, new StorageCredentials(accountName, accountKey));
// Set the service properties.
ServiceProperties sp = client.GetServiceProperties();
sp.DefaultServiceVersion = "2013-08-15";
CorsRule cr = new CorsRule();
cr.AllowedHeaders.Add("*");
cr.AllowedMethods = CorsHttpMethods.Get | CorsHttpMethods.Put;
cr.AllowedOrigins.Add("http://www.contoso.com");
cr.ExposedHeaders.Add("x-ms-*");
cr.MaxAgeInSeconds = 5;
sp.Cors.CorsRules.Clear();
sp.Cors.CorsRules.Add(cr);
client.SetServiceProperties(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 acknowledgement 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 acknowledgements 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 **CreateIfNotExists** 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, and Microsoft Message Analyzer 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](#)."

For more information about using Microsoft Message Analyzer, see "[Appendix 3: Using Microsoft Message Analyzer 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

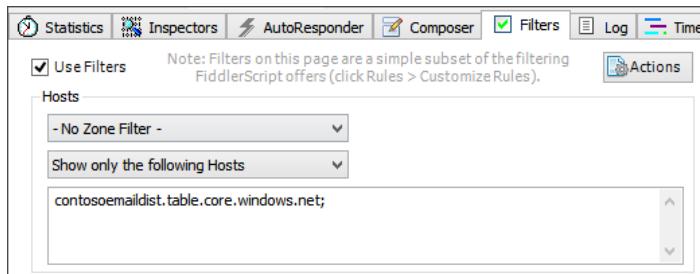
Fiddler can decode HTTPS traffic; you should read the Fiddler documentation carefully to understand how it does this, and to understand the security implications.

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:

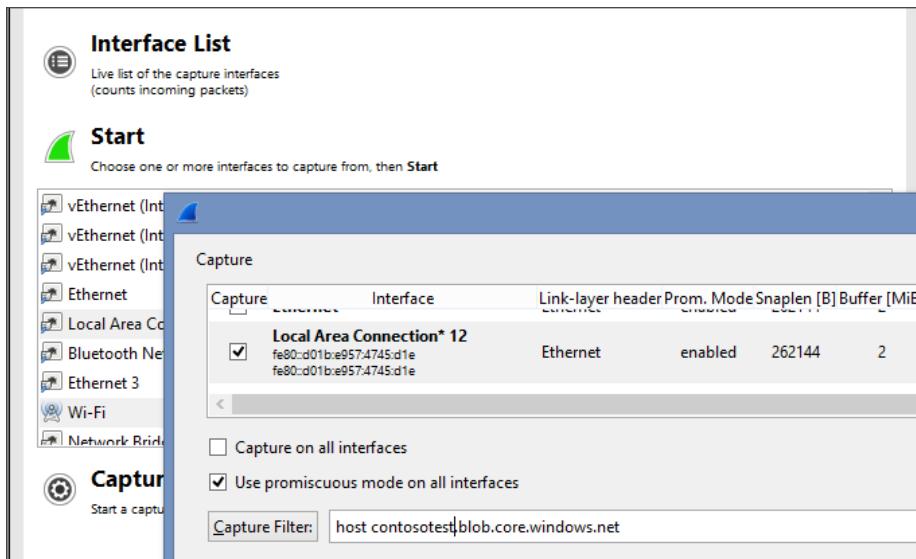


## Appendix 2: Using Wireshark to capture network traffic

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](#).

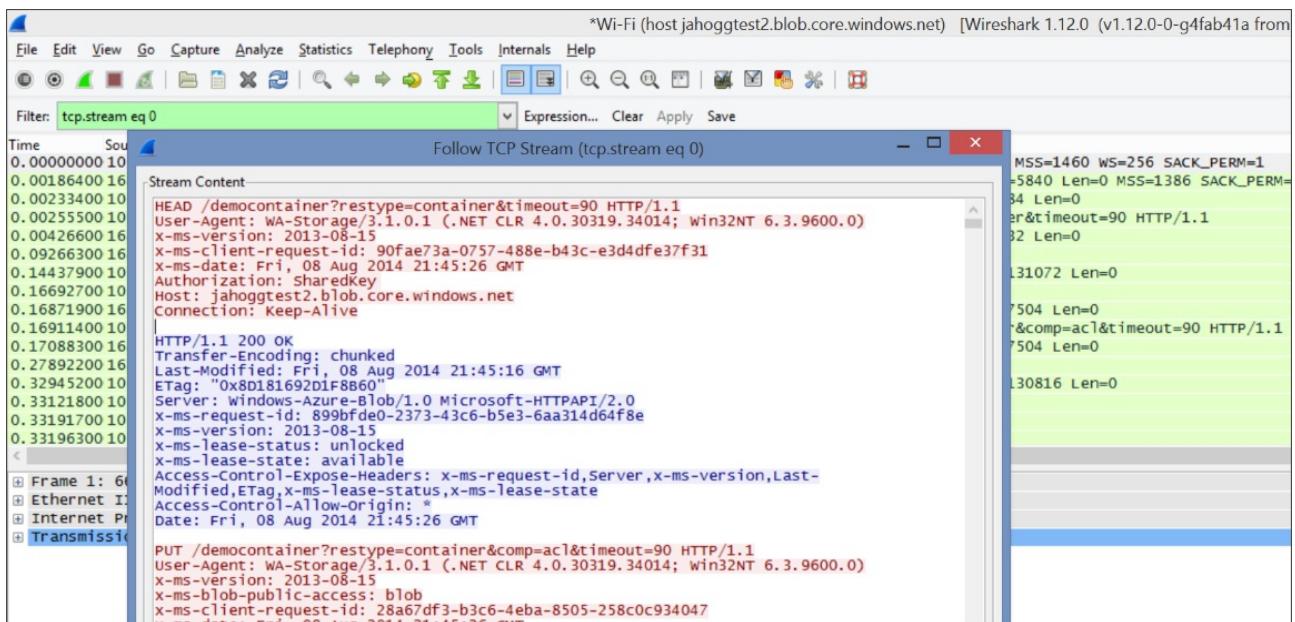


5. Click **Start**. Wireshark will now capture all the packets send to or from the table service endpoint as you use your client application on your local machine.
6. When you have finished, on the main menu click **Capture** and then **Stop**.
7. To save the captured data in a Wireshark Capture File, 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.

Errors: 3 (3)		Warnings: 5 (6)		Notes: 7 (32)		Chats: 13 (24)		Details: 65		Packet Comments: 0	
Group	Protocol	Summary								Count	
<input checked="" type="checkbox"/> Frame 8: 472 bytes on wire	Ethernet II, Src: 7c:7a:9	Bad checksum								1	
<input checked="" type="checkbox"/> Ethernet II, Src: 7c:7a:9	Internet Protocol Version 4 (IPv4)	Malformed Packet (Exception occurred)								1	
<input checked="" type="checkbox"/> Internet Protocol Version 4 (IPv4)	Transmission Control Protocol (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 3: Using Microsoft Message Analyzer to capture network traffic

You can use Microsoft Message Analyzer to capture HTTP and HTTPS traffic in a similar way to Fiddler, and capture network traffic in a similar way to Wireshark.

#### Configure a web tracing session using Microsoft Message Analyzer

To configure a web tracing session for HTTP and HTTPS traffic using Microsoft Message Analyzer, run the Microsoft Message Analyzer application and then on the File menu, click **Capture/Trace**. In the list of available trace scenarios, select **Web Proxy**. Then in the **Trace Scenario Configuration** panel, in the **HostnameFilter** textbox, add the names of your storage endpoints (you can look up these names in the [Azure portal](#)). For example, if the name of your Azure storage account is **contosodata**, you should add the following to the **HostnameFilter** textbox:

```
contosodata.blob.core.windows.net contosodata.table.core.windows.net contosodata.queue.core.windows.net
```

#### NOTE

A space character separates the hostnames.

When you are ready to start collecting trace data, click the **Start With** button.

For more information about the Microsoft Message Analyzer **Web Proxy** trace, see [Microsoft-Pef-WebProxy Provider](#).

The built-in **Web Proxy** trace in Microsoft Message Analyzer is based on Fiddler; it can capture client-side HTTPS traffic and display unencrypted HTTPS messages. The **Web Proxy** trace works by configuring a local proxy for all HTTP and HTTPS traffic that gives it access to unencrypted messages.

#### Diagnosing network issues using Microsoft Message Analyzer

In addition to using the Microsoft Message Analyzer **Web Proxy** trace to capture details of the HTTP/HTTPs traffic between the client application and the storage service, you can also use the built-in **Local Link Layer** trace to capture network packet information. This enables you to capture data similar to that which you can capture with Wireshark, and diagnose network issues such as dropped packets.

The following screenshot shows an example **Local Link Layer** trace with some informational messages in the **DiagnosisTypes** column. Clicking on an icon in the **DiagnosisTypes** column shows the details of the message. In this example, the server retransmitted message #305 because it did not receive an acknowledgement from the client:

MessageNumber	Type	Timestamp	TimeElapsed	Source
302	i	2014-06-27T11:09:05.5912057	0	192.168.0.16
303	i	2014-06-27T11:09:05.6019881	0	192.168.0.16
304	i	2014-06-27T11:09:05.6020185	0	192.168.0.16
305	i	2014-06-27T11:09:05.6270940	0	168.61.61.207
306	i	2014-06-27T11:09:05.6270947	0.0001652	168.61.61.207
307	i	2014-06-27T11:09:05.6271621	0	168.61.61.207

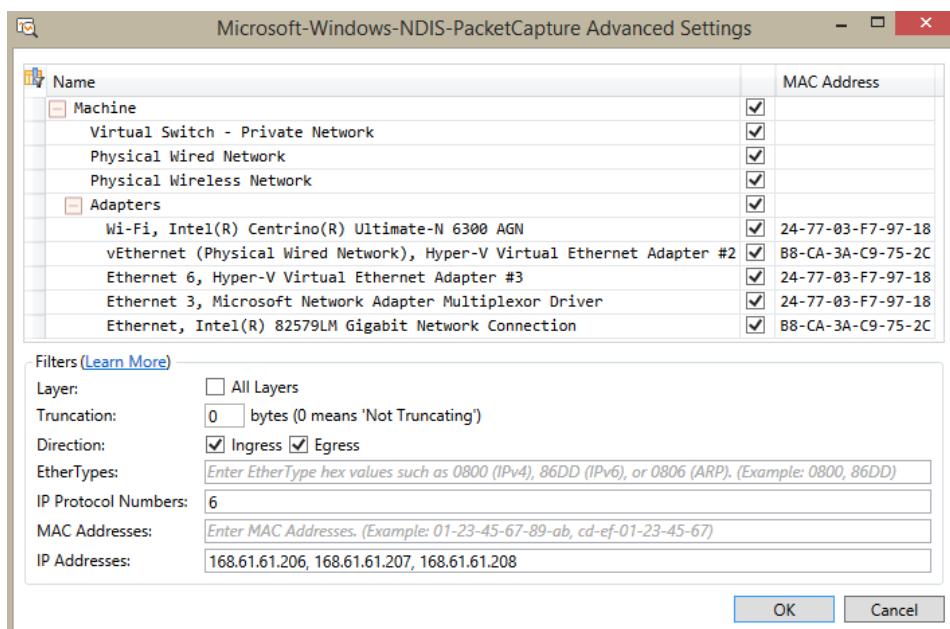
Type	Level	Message
i Application	Warning	TCP: Retransmitted, original message is #305.

Fields Stack Diagnosis

308	i	2014-06-27T11:09:05.6271629	0	168.61.61.207
310	i	2014-06-27T11:09:05.6272789	0	192.168.0.16

When you create the trace session in Microsoft Message Analyzer, you can specify filters to reduce the amount of noise in the trace. On the **Capture / Trace** page where you define the trace, click on the **Configure** link next to **Microsoft-Windows-NDIS-PacketCapture**. The following screenshot shows a configuration that filters TCP traffic for the IP addresses of three storage services:



For more information about the Microsoft Message Analyzer Local Link Layer trace, see [Microsoft-PER-NDIS-PacketCapture Provider](#).

#### 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](#)

# Monitor a storage account in the Azure portal

2/10/2020 • 6 minutes to read • [Edit Online](#)

Azure Storage Analytics provides metrics for all storage services, and logs for blobs, queues, and tables. You can use the [Azure portal](#) to configure which metrics and logs are recorded for your account, and configure charts that provide visual representations of your metrics data.

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.

## NOTE

There are costs associated with examining monitoring data in the Azure portal. For more information, see [Storage Analytics](#).

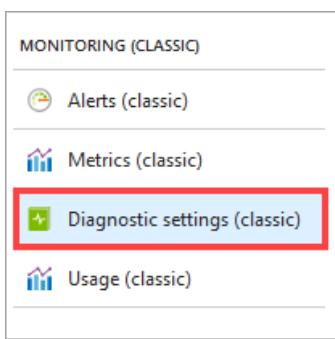
Azure Files currently supports Storage Analytics metrics, but does not yet support logging.

Premium performance block blob storage accounts don't support Storage Analytic metrics but they do support logging. You can enable logging programmatically via the REST API or the client library. If you want to view metrics with premium performance blob blob storage accounts, consider using [Azure Storage Metrics in Azure Monitor](#).

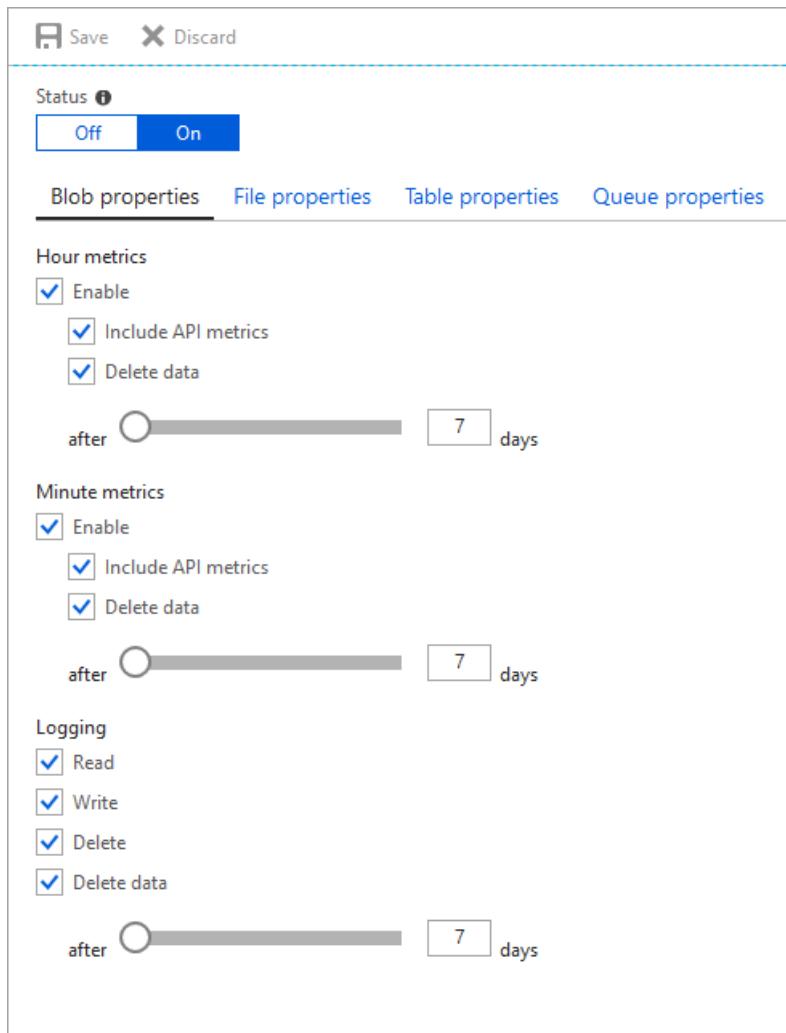
For an in-depth guide on using Storage Analytics and other tools to identify, diagnose, and troubleshoot Azure Storage-related issues, see [Monitor, diagnose, and troubleshoot Microsoft Azure Storage](#).

## Configure monitoring for a storage account

1. In the [Azure portal](#), select **Storage accounts**, then the storage account name to open the account dashboard.
2. Select **Diagnostics** in the MONITORING 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**.



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

You are charged when you manually delete metrics data. Stale analytics data (data older than your retention policy) is deleted by the system at no cost. We recommend setting a retention policy based on how long you want to retain storage analytics 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 storage account blade, as well as the individual service blades (blob, queue, table, and file). 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](#).

## Customize metrics charts

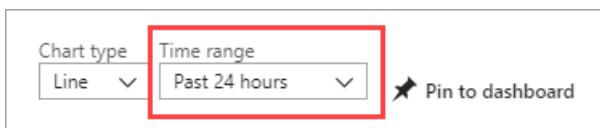
Use the following procedure to choose which storage metrics to view in a metrics chart.

- Start by displaying a storage metric chart in the Azure portal. You can find charts on the **storage account blade** and in the **Metrics** blade for an individual service (blob, queue, table).

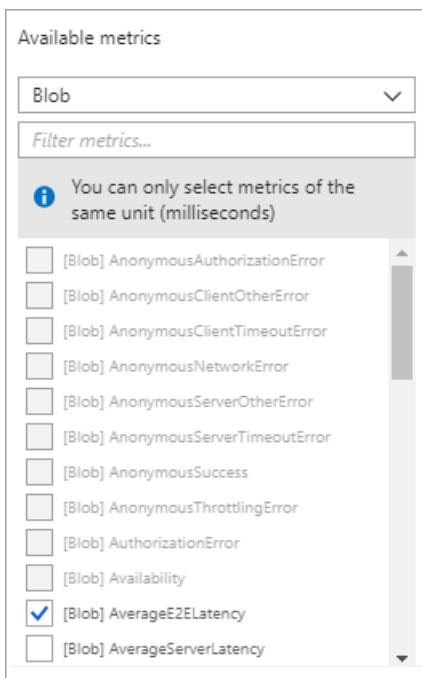
In this example, uses the following chart that appears on the **storage account blade**:



- Click anywhere within the chart to edit the chart.
- 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:



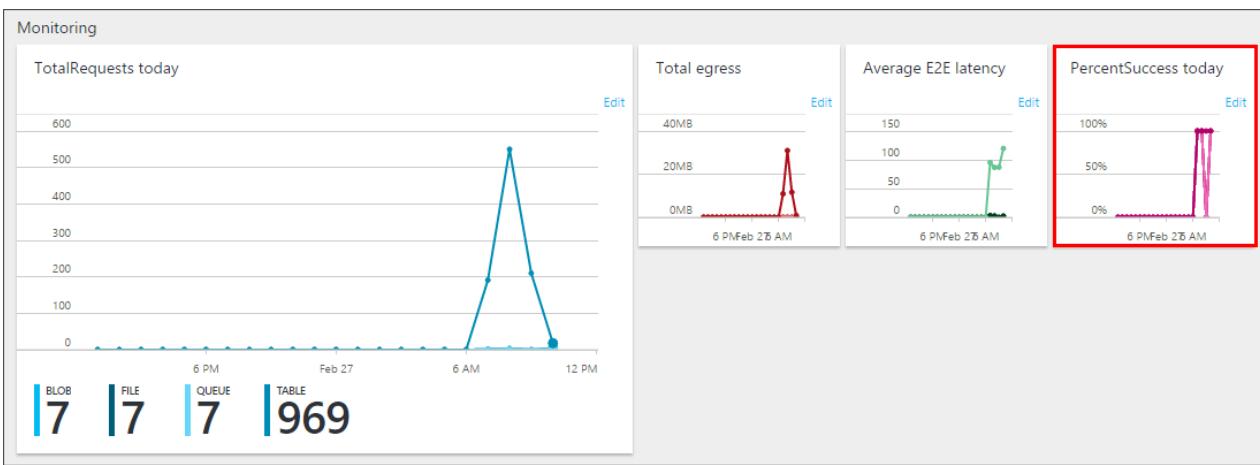
- Select the individual **metrics** you'd like displayed in the chart, then click **OK**.



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.

## Configure metrics alerts

You can create alerts to notify you when thresholds have been reached for storage resource metrics.

1. To open the **Alert rules blade**, scroll down to the **MONITORING** section of the **Menu blade** and select **Alerts (classic)**.
2. Select **Add metric alert (classic)** to open the **Add an alert rule** blade
3. Enter a **Name** and **Description** for your new alert rule.
4. Select the **Metric** for which you'd like to add an alert, an alert **Condition**, and a **Threshold**. The threshold unit type changes depending on the metric you've chosen. For example, "count" is the unit type for *ContainerCount*, while the unit for the *PercentNetworkError* metric is a percentage.
5. Select the **Period**. Metrics that reach or exceed the Threshold within the period trigger an alert.
6. (Optional) Configure **Email** and **Webhook** notifications. For more information on webhooks, see [Configure a webhook on an Azure metric alert](#). If you do not configure email or webhook notifications, alerts will appear only in the Azure portal.

**Add rule**

\* Name i  
myStorageAlertRule

Description  
Alert on Client Authorization Errors

Source  
Alert on

Metrics

Criteria  
Subscription  
My Subscription

Resource group  
myResourceGroup

Resource  
mymetricsdemo/blob

\* Metric [?](#)  
SASClientOtherError

2  
1.5  
No data to display  
0.5  
0  
7 PM

Condition  
Greater than or equal to

\* Threshold  
1

Period [?](#)  
Over the last 5 minutes

Notify via

Email owners, contributors, and readers

Additional administrator email(s)  
contoso@microsoft.com

Webhook [?](#)  
HTTP or HTTPS endpoint to route alerts to

[Learn more about configuring webhooks](#)

Take action [?](#) >  
Run a logic app from this alert

**OK**

The screenshot shows the 'Create alert rule' dialog for Azure Storage metrics. The 'Resource' is set to 'mymetricsdemo/blob' and the 'Metric' is 'SASClientOtherError'. The chart shows 'No data to display' for the selected period. The 'Condition' is set to 'Greater than or equal to 1' over 'Over the last 5 minutes'. Under 'Notify via', 'Email owners, contributors, and readers' is checked, and an additional administrator email 'contoso@microsoft.com' is listed. A webhook endpoint is also specified. Action options include 'Run a logic app from this alert'.

## 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.

The screenshot shows the 'Tile Gallery' interface in the Azure portal. It has filters for 'Type' (Storage accounts), 'Resources' (mystorageaccount), and 'Categories' (Monitoring). A red box highlights a chart titled 'StorageAccount - Total request..'. Below it are other charts for 'StorageAccount - Average E2E ...' and 'StorageAccount - Availability'.

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](#).

## Configure logging

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 yet support logging.

1. In the [Azure portal](#), select **Storage accounts**, then the name of the storage account to open the storage account blade.
2. Select **Diagnostics settings (classic)** in the **Monitoring (classic)** section of the menu blade.

The screenshot shows the 'MONITORING (CLASSIC)' section of the storage account blade. It includes options for 'Alerts (classic)', 'Metrics (classic)', 'Diagnostic settings (classic)' (which is highlighted with a red box), and 'Usage (classic)'.

3. Ensure Status is set to **On**, and select the **services** for which you'd like to enable logging.

The screenshot shows the 'File properties' tab of the Azure Storage Analytics configuration. The 'Status' is set to 'On'. Under 'Hour metrics', 'Enable' is checked, along with 'Include API metrics' and 'Delete data'. A progress bar indicates the data will be retained for 7 days. Under 'Minute metrics', 'Enable' is checked, along with 'Include API metrics' and 'Delete data'. A progress bar indicates the data will be retained for 7 days. Under 'Logging', 'Read' is checked, along with 'Write', 'Delete', and 'Delete data'. A progress bar indicates the data will be retained for 7 days. At the top right, there are 'Save' and 'Discard' buttons.

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 Storage Explorer](#), or programmatically using the storage client library or PowerShell.

For information about accessing the `$logs` container, see [Storage analytics logging](#).

## Next steps

- Find more details about [metrics, logging, and billing](#) for Storage Analytics.

# Azure Storage metrics in Azure Monitor

3/25/2020 • 13 minutes to read • [Edit Online](#)

With metrics on Azure Storage, you can analyze usage trends, trace requests, and diagnose issues with your storage account.

Azure Monitor provides unified user interfaces for monitoring across different Azure services. For more information, see [Azure Monitor](#). Azure Storage integrates Azure Monitor by sending metric data to the Azure Monitor platform.

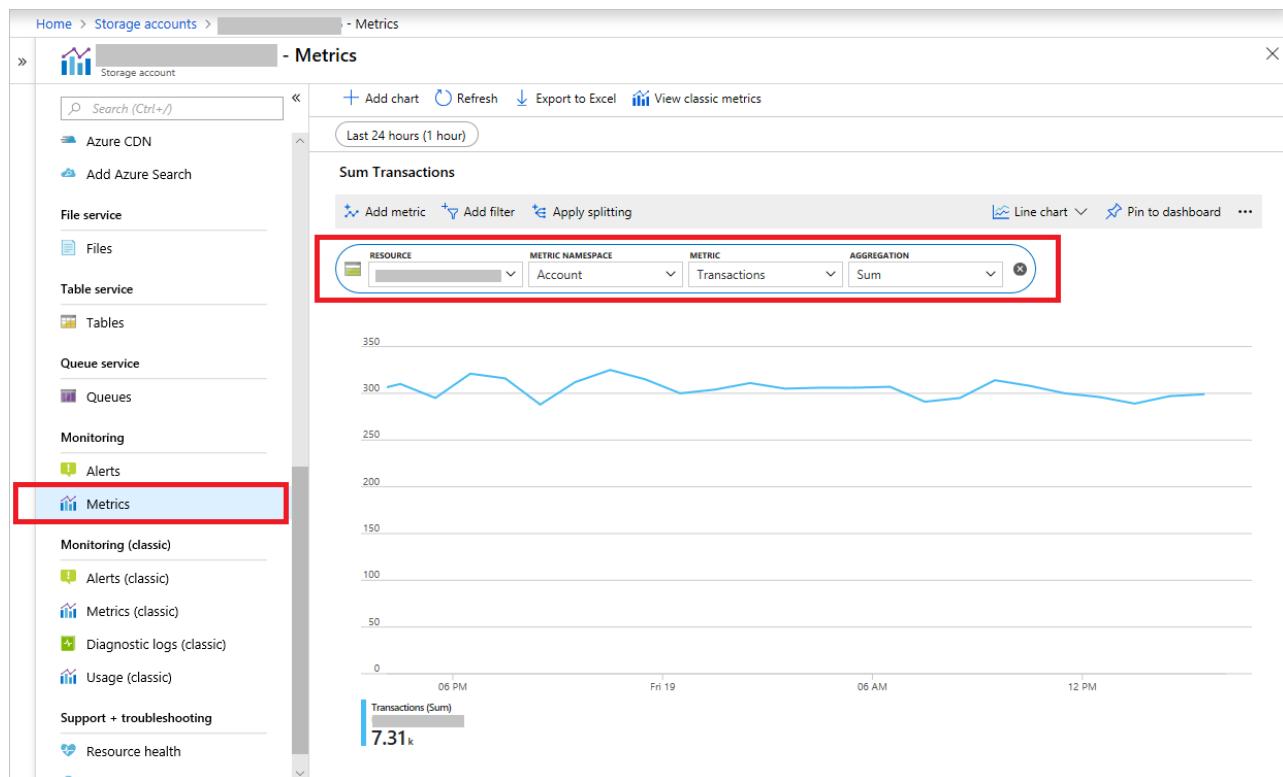
## Access metrics

Azure Monitor provides multiple ways to access metrics. You can access them from the [Azure portal](#), the Azure Monitor APIs (REST, and .NET) and analysis solutions such as Event Hubs. For more information, see [Azure Monitor Metrics](#).

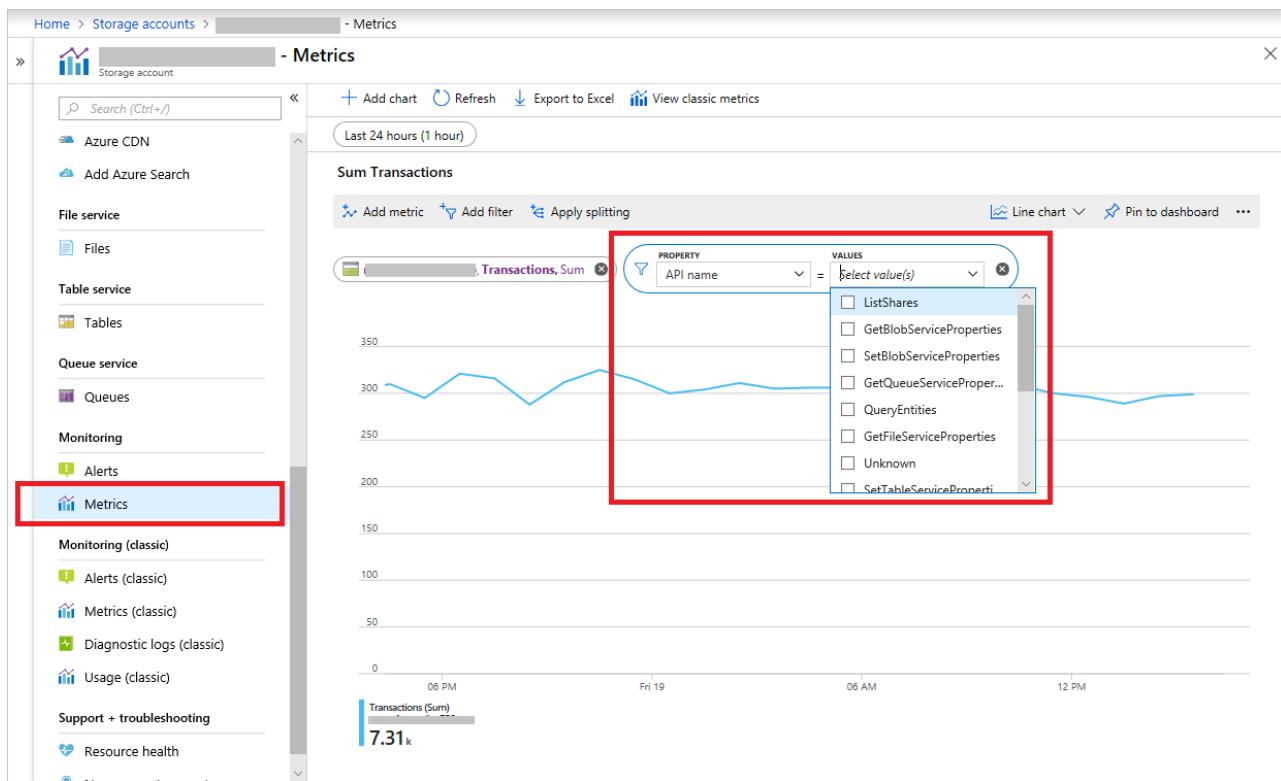
Metrics are enabled by default, and you can access the past 93 days of data. If you need to retain data for a longer period of time, you can archive metrics data to an Azure Storage account. This is configured in [diagnostic settings](#) in Azure Monitor.

### Access metrics in the Azure portal

You can monitor metrics over time in the Azure portal. The following example shows how to view **Transactions** at account level.



For metrics supporting dimensions, you can filter metric with the desired dimension value. The following example shows how to view **Transactions** at account level on a specific operation by selecting values for **API Name** dimension.



## Access metrics with the REST API

Azure Monitor provides [REST APIs](#) to read metric definition and values. This section shows you how to read the storage metrics. Resource ID is used in all REST APIs. For more information, please read [Understanding resource ID for services in Storage](#).

The following example shows how to use [ArmClient](#) at the command line to simplify testing with the REST API.

### List account level metric definition with the REST API

The following example shows how to list metric definition at account level:

```
Login to Azure and enter your credentials when prompted.
> armclient login

> armclient GET
/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/
{storageAccountName}/providers/microsoft.insights/metricdefinitions?api-version=2018-01-01
```

If you want to list the metric definitions for blob, table, file, or queue, you must specify different resource IDs for each service with the API.

The response contains the metric definition in JSON format:

```
{
 "value": [
 {
 "id": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/{storageAccountName}/providers/microsoft.insights/metricdefinitions/UsedCapacity",
 "resourceId": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/{storageAccountName}",
 "category": "Capacity",
 "name": {
 "value": "UsedCapacity",
 "localizedValue": "Used capacity"
 },
 "isDimensionRequired": false,
 "unit": "Bytes",
 "primaryAggregationType": "Average",
 "metricAvailabilities": [
 {
 "timeGrain": "PT1M",
 "retention": "P30D"
 },
 {
 "timeGrain": "PT1H",
 "retention": "P30D"
 }
],
 ...
 ... next metric definition
]
 }
}
```

#### **Read account-level metric values with the REST API**

The following example shows how to read metric data at account level:

```
> armclient GET
"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/{storageAccountName}/providers/microsoft.insights/metrics?metricnames=Availability&api-version=2018-01-01&aggregation=Average&interval=PT1H"
```

In above example, if you want to read metric values for blob, table, file, or queue, you must specify different resource IDs for each service with the API.

The following response contains metric values in JSON format:

```
{
 "cost": 0,
 "timespan": "2017-09-07T17:27:41Z/2017-09-07T18:27:41Z",
 "interval": "PT1H",
 "value": [
 {
 "id":
 "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts
 /{storageAccountName}/providers/Microsoft.Insights/metrics/Availability",
 "type": "Microsoft.Insights/metrics",
 "name": {
 "value": "Availability",
 "localizedValue": "Availability"
 },
 "unit": "Percent",
 "timeseries": [
 {
 "metadatavalues": [],
 "data": [
 {
 "timeStamp": "2017-09-07T17:27:00Z",
 "average": 100.0
 }
]
 }
]
 }
]
}
```

## Access metrics with the .NET SDK

Azure Monitor provides [.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 later version for storage metrics. Resource ID is used in .NET SDK. For more information, please read [Understanding resource ID for services in Storage](#).

The following example shows how to use Azure Monitor .NET SDK to read storage metrics.

### List account level metric definition with the .NET SDK

The following example shows how to list metric definition at account level:

```

public static async Task ListStorageMetricDefinition()
{
 // Resource ID for storage account
 var resourceId =
"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts
/{storageAccountName}";
 var subscriptionId = "{SubscriptionID}";
 // How to identify Tenant ID, Application ID and Access Key:
https://azure.microsoft.com/documentation/articles/resource-group-create-service-principal-portal/
 var tenantId = "{TenantID}";
 var applicationId = "{ApplicationID}";
 var accessKey = "{AccessKey}";

 // Using metrics in Azure Monitor is currently free. However, if you use additional solutions ingesting
 metrics data, you may be billed by these solutions. For example, you are billed by Azure Storage if you archive
 metrics data to an Azure Storage account. Or you are billed by Operation Management Suite (OMS) if you stream
 metrics data to OMS for advanced analysis.

 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)
 {
 //Enumrate metric definition:
 // Id
 // ResourceId
 // Name
 // Unit
 // MetricAvailabilities
 // PrimaryAggregationType
 // Dimensions
 // IsDimensionRequired
 }
}

```

If you want to list the metric definitions for blob, table, file, or queue, you must specify different resource IDs for each service with the API.

#### **Read metric values with the .NET SDK**

The following example shows how to read `UsedCapacity` data at account level:

```

public static async Task ReadStorageMetricValue()
{
 // Resource ID for storage account
 var resourceId =
"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts
/{storageAccountName}";
 var subscriptionId = "{SubscriptionID}";
 // How to identify Tenant ID, Application ID and Access Key:
https://azure.microsoft.com/documentation/articles/resource-group-create-service-principal-portal/
 var tenantId = "{TenantID}";
 var applicationId = "{ApplicationID}";
 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)
 {
 //Enumrate metric value
 // Id
 // Name
 // Type
 // Unit
 // Timeseries
 // - Data
 // - Metadatavalues
 }
}

```

In above example, if you want to read metric values for blob, table, file, or queue, you must specify different resource IDs for each service with the API.

#### **Read multi-dimensional metric values with the .NET SDK**

For multi-dimensional metrics, you need to define meta data filter if you want to read metric data on specific dimension value.

The following example shows how to read metric data on the metric supporting multi-dimension:

```

public static async Task ReadStorageMetricValueTest()
{
 // Resource ID for blob storage
 var resourceId =
"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts
/{storageAccountName}/blobServices/default";
 var subscriptionId = "{SubscriptionID}";
 // How to identify Tenant ID, Application ID and Access Key:
https://azure.microsoft.com/documentation/articles/resource-group-create-service-principal-portal/
 var tenantId = "{TenantID}";
 var applicationId = "{ApplicationID}";
 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)
 {
 //Enumrate metric value
 // Id
 // Name
 // Type
 // Unit
 // Timeseries
 // - Data
 // - Metadatavalues
 }
}

```

## Understanding resource ID for services in Azure Storage

Resource ID is a unique identifier of a resource in Azure. When you use the Azure Monitor REST API to read metrics definitions or values, you must use resource ID for the resource on which you intend to operate. The resource ID template follows this format:

```
/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/{resourceProviderNamespace}/{resou
rceType}/{resourceName}
```

Storage provides metrics at both the storage account level and the service level with Azure Monitor. For example, you can retrieve metrics for just Blob storage. Each level has its own resource ID, which is used to retrieve the metrics for just that level.

## Resource ID for a storage account

The following shows the format for specifying the Resource ID for a storage account.

```
/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/{storageAccountName}
```

## Resource ID for the storage services

The following shows the format for specifying the Resource ID for each of the storage services.

- Blob service resource ID

```
/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/{storageAccountName}/blobServices/default
```

- Table service resource ID

```
/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/{storageAccountName}/tableServices/default
```

- Queue service resource ID

```
/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/{storageAccountName}/queueServices/default
```

- File service resource ID

```
/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/{storageAccountName}/fileServices/default
```

## Resource ID in Azure Monitor REST API

The following shows the pattern used when calling the Azure Monitor REST API.

```
GET {resourceId}/providers/microsoft.insights/metrics?{parameters}
```

## Capacity metrics

Capacity metrics values are sent to Azure Monitor every hour. The values are refreshed daily. 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

METRIC NAME	DESCRIPTION
-------------	-------------

METRIC NAME	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

METRIC NAME	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="#">BlobTier</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="#">BlobTier</a> (<a href="#">Definition</a>)</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>

## Table storage

METRIC NAME	DESCRIPTION
TableCapacity	<p>The amount of Table storage used by the storage account.</p> <p>Unit: Bytes Aggregation Type: Average Value example: 1024</p>
TableCount	<p>The number of tables in the storage account.</p> <p>Unit: Count Aggregation Type: Average Value example: 1024</p>

METRIC NAME	DESCRIPTION
TableEntityCount	<p>The number of table entities in the storage account.</p> <p>Unit: Count Aggregation Type: Average Value example: 1024</p>

## Queue storage

METRIC NAME	DESCRIPTION
QueueCapacity	<p>The amount of Queue storage used by the storage account.</p> <p>Unit: Bytes Aggregation Type: Average Value example: 1024</p>
QueueCount	<p>The number of queues in the storage account.</p> <p>Unit: Count Aggregation Type: Average Value example: 1024</p>
QueueMessageCount	<p>The number of unexpired queue messages in the storage account.</p> <p>Unit: Count Aggregation Type: Average Value example: 1024</p>

## File storage

METRIC NAME	DESCRIPTION
FileCapacity	<p>The amount of File storage used by the storage account.</p> <p>Unit: Bytes Aggregation Type: Average Value example: 1024</p>
FileCount	<p>The number of files in the storage account.</p> <p>Unit: Count Aggregation Type: Average Value example: 1024</p>
FileShareCount	<p>The number of file shares in the storage account.</p> <p>Unit: Count 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 service level (Blob storage, Table storage, Azure Files, and Queue storage).

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 NAME	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: ResponseType, GeoType, ApiName, and Authentication (<a href="#">Definition</a>) Value example: 1024</p>
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 to an external client from 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.</p> <p>Unit: Milliseconds Aggregation Type: Average Applicable dimensions: GeoType, ApiName, and Authentication (<a href="#">Definition</a>) Value example: 1024</p>

METRIC NAME	DESCRIPTION
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.

DIMENSION NAME	DESCRIPTION
<b>BlobType</b>	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 Blob is included in BlockBlob.
<b>BlobTier</b>	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>• <b>Hot</b>: Hot tier</li> <li>• <b>Cool</b>: Cool tier</li> <li>• <b>Archive</b>: Archive tier</li> <li>• <b>Premium</b>: Premium tier for block blob</li> <li>• <b>P4/P6/P10/P15/P20/P30/P40/P50/P60</b>: Tier types for premium page blob</li> <li>• <b>Standard</b>: Tier type for standard page Blob</li> <li>• <b>Untiered</b>: Tier type for general purpose v1 storage account</li> </ul>
<b>GeoType</b>	Transaction from Primary or Secondary cluster. The available values include <b>Primary</b> and <b>Secondary</b> . 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>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>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> </ul>
ApiName	<p>The name of operation. For example:</p> <ul style="list-style-type: none"> <li>• <b>CreateContainer</b></li> <li>• <b>DeleteBlob</b></li> <li>• <b>GetBlob</b></li> </ul> <p>For all operation names, see <a href="#">document</a>.</p>
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>

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**. Or if you look at **BlobCount** value for Block Blob, you need to filter the **BlobType** dimension with **BlockBlob**.

## Service continuity of legacy metrics

Legacy metrics are available in parallel with Azure Monitor managed metrics. The support keeps the same until Azure Storage ends the service on legacy metrics.

## FAQ

### Does new metrics support Classic Storage account?

No, new metrics in Azure Monitor only support Azure Resource Manager storage accounts. If you want to use metrics on Storage accounts, you need to migrate to Azure Resource Manager Storage account. See [Migrate to Azure Resource Manager](#).

### Does Azure Storage support metrics for Managed Disks or Unmanaged Disks?

No, Azure Compute supports the metrics on disks. See [article](#) for more details.

### How to map and migrate classic metrics with new metrics?

You can find detailed mapping between classic metrics and new metrics in [Azure Storage metrics migration](#).

## Next steps

- [Azure Monitor](#)

# Azure Storage metrics migration

8/7/2019 • 4 minutes to read • [Edit Online](#)

Aligned with the strategy of unifying the monitor experience in Azure, Azure Storage integrates metrics to the Azure Monitor platform. In the future, the service of the old metrics will end with an early notification based on Azure policy. If you rely on old storage metrics, you need to migrate prior to the service end date in order to maintain your metric information.

This article shows you how to migrate from the old metrics to the new metrics.

## Understand old metrics that are managed by Azure Storage

Azure Storage collects old metric values, and aggregates and stores them in \$Metric tables within the same storage account. You can use the Azure portal to set up a monitoring chart. You can also use the Azure Storage SDKs to read the data from \$Metric tables that are based on the schema. For more information, see [Storage Analytics](#).

Old metrics provide capacity metrics only on Azure Blob storage. Old metrics provide transaction metrics on Blob storage, Table storage, Azure Files, and Queue storage.

Old metrics are designed in a flat schema. The design results in zero metric value when you don't have the traffic patterns triggering the metric. For example, the **ServerTimeoutError** value is set to 0 in \$Metric tables even when you don't receive any server timeout errors from the live traffic to a storage account.

## Understand new metrics managed by Azure Monitor

For new storage metrics, Azure Storage emits the metric data to the Azure Monitor back end. Azure Monitor provides a unified monitoring experience, including data from the portal as well as data ingestion. For more details, you can refer to this [article](#).

New metrics provide capacity metrics and transaction metrics on Blob, Table, File, Queue, and premium storage.

Multi-dimension is one of the features that Azure Monitor provides. Azure Storage adopts the design in defining new metric schema. For supported dimensions on metrics, you can find details in [Azure Storage metrics in Azure Monitor](#). Multi-dimension design provides cost efficiency on both bandwidth from ingestion and capacity from storing metrics. Consequently, if your traffic has not triggered related metrics, the related metric data will not be generated. For example, if your traffic has not triggered any server timeout errors, Azure Monitor doesn't return any data when you query the value of metric **Transactions** with dimension **ResponseType** equal to **ServerTimeoutError**.

## Metrics mapping between old metrics and new metrics

If you read metric data programmatically, you need to adopt the new metric schema in your programs. To better understand the changes, you can refer to the mapping listed in the following table:

### Capacity metrics

OLD METRIC	NEW METRIC
Capacity	<b>BlobCapacity</b> with the dimension <b>BlobType</b> equal to <b>BlockBlob</b> or <b>PageBlob</b>

OLD METRIC	NEW METRIC
ObjectCount	BlobCount with the dimension <b>BlobType</b> equal to <b>BlockBlob</b> or <b>PageBlob</b>
ContainerCount	ContainerCount

The following metrics are new offerings that the old metrics don't support:

- **TableCapacity**
- **TableCount**
- **TableEntityCount**
- **QueueCapacity**
- **QueueCount**
- **QueueMessageCount**
- **FileCapacity**
- **FileCount**
- **FileShareCount**
- **UsedCapacity**

#### Transaction metrics

OLD METRIC	NEW METRIC
<b>AnonymousAuthorizationError</b>	Transactions with the dimension <b>ResponseType</b> equal to <b>AuthorizationError</b> and dimension <b>Authentication</b> equal to <b>Anonymous</b>
<b>AnonymousClientOtherError</b>	Transactions with the dimension <b>ResponseType</b> equal to <b>ClientOtherError</b> and dimension <b>Authentication</b> equal to <b>Anonymous</b>
<b>AnonymousClientTimeoutError</b>	Transactions with the dimension <b>ResponseType</b> equal to <b>ClientTimeoutError</b> and dimension <b>Authentication</b> equal to <b>Anonymous</b>
<b>AnonymousNetworkError</b>	Transactions with the dimension <b>ResponseType</b> equal to <b>NetworkError</b> and dimension <b>Authentication</b> equal to <b>Anonymous</b>
<b>AnonymousServerOtherError</b>	Transactions with the dimension <b>ResponseType</b> equal to <b>ServerOtherError</b> and dimension <b>Authentication</b> equal to <b>Anonymous</b>
<b>AnonymousServerTimeoutError</b>	Transactions with the dimension <b>ResponseType</b> equal to <b>ServerTimeoutError</b> and dimension <b>Authentication</b> equal to <b>Anonymous</b>
<b>AnonymousSuccess</b>	Transactions with the dimension <b>ResponseType</b> equal to <b>Success</b> and dimension <b>Authentication</b> equal to <b>Anonymous</b>
<b>AnonymousThrottlingError</b>	Transactions with the dimension <b>ResponseType</b> equal to <b>ClientThrottlingError</b> or <b>ServerBusyError</b> and dimension <b>Authentication</b> equal to <b>Anonymous</b>

OLD METRIC	NEW METRIC
AuthorizationError	Transactions with the dimension <b>ResponseType</b> equal to <b>AuthorizationError</b>
Availability	Availability
AverageE2ELatency	SuccessE2ELatency
AverageServerLatency	SuccessServerLatency
ClientOtherError	Transactions with the dimension <b>ResponseType</b> equal to <b>ClientOtherError</b>
ClientTimeoutError	Transactions with the dimension <b>ResponseType</b> equal to <b>ClientTimeoutError</b>
NetworkError	Transactions with the dimension <b>ResponseType</b> equal to <b>NetworkError</b>
PercentAuthorizationError	Transactions with the dimension <b>ResponseType</b> equal to <b>AuthorizationError</b>
PercentClientOtherError	Transactions with the dimension <b>ResponseType</b> equal to <b>ClientOtherError</b>
PercentNetworkError	Transactions with the dimension <b>ResponseType</b> equal to <b>NetworkError</b>
PercentServerOtherError	Transactions with the dimension <b>ResponseType</b> equal to <b>ServerOtherError</b>
PercentSuccess	Transactions with the dimension <b>ResponseType</b> equal to <b>Success</b>
PercentThrottlingError	Transactions with the dimension <b>ResponseType</b> equal to <b>ClientThrottlingError</b> or <b>ServerBusyError</b>
PercentTimeoutError	Transactions with the dimension <b>ResponseType</b> equal to <b>ServerTimeoutError</b> or <b>ResponseType</b> equal to <b>ClientTimeoutError</b>
SASAuthorizationError	Transactions with the dimension <b>ResponseType</b> equal to <b>AuthorizationError</b> and dimension <b>Authentication</b> equal to <b>SAS</b>
SASClientOtherError	Transactions with the dimension <b>ResponseType</b> equal to <b>ClientOtherError</b> and dimension <b>Authentication</b> equal to <b>SAS</b>
SASClientTimeoutError	Transactions with the dimension <b>ResponseType</b> equal to <b>ClientTimeoutError</b> and dimension <b>Authentication</b> equal to <b>SAS</b>
SASNetworkError	Transactions with the dimension <b>ResponseType</b> equal to <b>NetworkError</b> and dimension <b>Authentication</b> equal to <b>SAS</b>

OLD METRIC	NEW METRIC
SASServerOtherError	Transactions with the dimension <b>ResponseType</b> equal to <b>ServerOtherError</b> and dimension <b>Authentication</b> equal to <b>SAS</b>
SASServerTimeoutError	Transactions with the dimension <b>ResponseType</b> equal to <b>ServerTimeoutError</b> and dimension <b>Authentication</b> equal to <b>SAS</b>
SASSuccess	Transactions with the dimension <b>ResponseType</b> equal to <b>Success</b> and dimension <b>Authentication</b> equal to <b>SAS</b>
SASThrottlingError	Transactions with the dimension <b>ResponseType</b> equal to <b>ClientThrottlingError</b> or <b>ServerBusyError</b> and dimension <b>Authentication</b> equal to <b>SAS</b>
ServerOtherError	Transactions with the dimension <b>ResponseType</b> equal to <b>ServerOtherError</b>
ServerTimeoutError	Transactions with the dimension <b>ResponseType</b> equal to <b>ServerTimeoutError</b>
Success	Transactions with the dimension <b>ResponseType</b> equal to <b>Success</b>
ThrottlingError	Transactions with the dimension <b>ResponseType</b> equal to <b>ClientThrottlingError</b> or <b>ServerBusyError</b>
TotalBillableRequests	Transactions
TotalEgress	Egress
TotalIngress	Ingress
TotalRequests	Transactions

## FAQ

### How should I migrate existing alert rules?

If you have created classic alert rules based on old storage metrics, you need to create new alert rules based on the new metric schema.

### Is new metric data stored in the same storage account by default?

No. To archive the metric data to a storage account, use the [Azure Monitor Diagnostic Setting API](#).

## Next steps

- [Azure Monitor](#)
- [Storage metrics in Azure Monitor](#)

# Storage Analytics

4/3/2020 • 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 are not charged for delete transactions when Storage Analytics deletes old logging and metrics data. However, delete transactions from a client are billable. 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)

10/16/2019 • 12 minutes to read • [Edit Online](#)

Storage Analytics can store metrics that include aggregated transaction statistics and capacity data about requests to a storage service. Transactions are reported at both the API operation level as well as at the storage service level, and 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, and to 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](#); 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 Service Properties operations to enable Storage Analytics for each service.

## NOTE

Storage Analytics metrics are available for the Blob, Queue, Table, and File services. Storage Analytics metrics are now Classic metrics. Microsoft recommends using [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, including ingress/egress, availability, errors, and categorized request percentages. You can see a complete list of the transaction details in the [Storage Analytics Metrics Table Schema](#) topic.

Transaction data is recorded at two levels – the service level and the API operation level. At the service level, statistics summarizing 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 will log the request and include it in the aggregated data for both the Blob service as well as the **GetBlob** operation. However, if no **GetBlob** operation is requested during the hour, an entity will not be written to the `$MetricsTransactionsBlob` for that operation.

Transaction metrics are recorded for both 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 only available 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 the 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 for transaction information, one table for minute transaction information, and another table for capacity information. Transaction and minute transaction information consists of request and response data, and capacity information consists of storage usage data. Hour metrics, minute metrics, and capacity for a storage account's Blob service is can be accessed in tables that are named as described in the following table.

METRICS LEVEL	TABLE NAMES	SUPPORTED FOR VERSIONS
Hourly metrics, primary location	- \$MetricsTransactionsBlob - \$MetricsTransactionsTable - \$MetricsTransactionsQueue	Versions prior to 2013-08-15 only. While these names are still supported, it's recommended that you switch to using the tables listed below.
Hourly metrics, primary location	- \$MetricsHourPrimaryTransactionsBlob - \$MetricsHourPrimaryTransactionsTable - \$MetricsHourPrimaryTransactionsQueue - \$MetricsHourPrimaryTransactionsFile	All versions. Support for File service metrics is available only in version 2015-04-05 and later.
Minute metrics, primary location	- \$MetricsMinutePrimaryTransactionsBlob - \$MetricsMinutePrimaryTransactionsTable - \$MetricsMinutePrimaryTransactionsQueue - \$MetricsMinutePrimaryTransactionsFile	All versions. Support for File service metrics is available only in version 2015-04-05 and later.
Hourly metrics, secondary location	- \$MetricsHourSecondaryTransactionsBlob - \$MetricsHourSecondaryTransactionsTable - \$MetricsHourSecondaryTransactionsQueue	All versions. Read-access geo-redundant replication must be enabled.

METRICS LEVEL	TABLE NAMES	SUPPORTED FOR VERSIONS
Minute metrics, secondary location	- \$MetricsMinuteSecondaryTransactionsBlob - \$MetricsMinuteSecondaryTransactionsTable - \$MetricsMinuteSecondaryTransactionsQueue	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 are accessed via the namespace of the storage account, for example:

`https://<accountname>.table.core.windows.net/Tables("$MetricsTransactionsBlob")`. The metrics tables do not appear in a listing operation, and must be accessed directly via the table name.

## Enable metrics using the Azure portal

Follow these steps to enable metrics in the [Azure portal](#):

1. Navigate to your storage account.
2. Select **Diagnostics settings (classic)** from the **Menu** pane.
3. Ensure that **Status** is set to **On**.
4. Select the metrics for the services you wish to monitor.
5. Specify a retention policy to indicate how long to retain metrics and log data.
6. Select **Save**.

The [Azure portal](#) does not currently enable you to configure minute metrics in your storage account; you must enable minute metrics using PowerShell or programmatically.

## Enable Storage metrics using PowerShell

You can use PowerShell on your local machine to configure Storage Metrics in your storage account by using the Azure PowerShell cmdlet **Get-AzStorageServiceMetricsProperty** to retrieve the current settings, and the cmdlet **Set-AzStorageServiceMetricsProperty** to change the current settings.

The cmdlets that control Storage Metrics use the following parameters:

- **ServiceType**, possible value are **Blob**, **Queue**, **Table**, and **File**.
- **MetricsType**, possible values are **Hour** and **Minute**.
- **MetricsLevel**, possible values are:
- **None**: Turns off monitoring.
- **Service**: Collects metrics such as ingress/egress, availability, latency, and success percentages, which are aggregated for the blob, queue, table, and file services.
- **ServiceAndApi**: In addition to the Service metrics, collects the same set of metrics for each storage operation in the Azure Storage service API.

For example, the following command switches on minute metrics for the blob service in your storage account with the retention period set to five days:

#### NOTE

This command assumes that you've signed into your Azure subscription by using the `Connect-AzAccount` command.

```
$storageAccount = Get-AzStorageAccount -ResourceGroupName "<resource-group-name>" -AccountName "<storage-account-name>"

Set-AzStorageServiceMetricsProperty -MetricsType Minute -ServiceType Blob -MetricsLevel ServiceAndApi -
RetentionDays 5 -Context $storageAccount.Context
```

- Replace the `<resource-group-name>` placeholder value with the name of your resource group.
- Replace the `<storage-account-name>` placeholder value with the name of your storage account.

The following command retrieves the current hourly metrics level and retention days for the blob service in your default storage account:

```
Get-AzStorageServiceMetricsProperty -MetricsType Hour -ServiceType Blob -Context $storagecontext.Context
```

For information about how to configure the Azure PowerShell cmdlets to work with your Azure subscription and how to select the default storage account to use, see: [How to install and configure Azure PowerShell](#).

## Enable Storage metrics programmatically

In addition to using the Azure portal or the Azure PowerShell cmdlets to control Storage Metrics, you can also use one of the Azure Storage APIs. For example, if you are using a .NET language you can use the Storage Client Library.

The classes `CloudBlobClient`, `CloudQueueClient`, `CloudTableClient`, and `CloudFileClient` all have methods such as `SetServiceProperties` and `SetServicePropertiesAsync` that take a `ServiceProperties` object as a parameter. You can use the `ServiceProperties` object to configure Storage Metrics. For example, the following C# snippet shows how to change the metrics level and retention days for the hourly queue metrics:

```
var storageAccount = CloudStorageAccount.Parse(connStr);
var queueClient = storageAccount.CreateCloudQueueClient();
var serviceProperties = queueClient.GetServiceProperties();

serviceProperties.HourMetrics.MetricsLevel = MetricsLevel.Service;
serviceProperties.HourMetrics.RetentionDays = 10;

queueClient.SetServiceProperties(serviceProperties);
```

For more information about using a .NET language to configure Storage Metrics, see [Storage Client Library for .NET](#).

For general information about configuring Storage Metrics using the REST API, see [Enabling and Configuring Storage Analytics](#).

## Viewing Storage metrics

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](#):

1. Navigate to your storage account in the [Azure portal](#).

2. Select **Metrics (classic)** in the **Menu** blade for the service whose metrics you want to view.
3. Click the chart you want to configure.
4. In the **Edit Chart** blade, select the **Time Range**, **Chart type**, and the metrics you want displayed in the chart.

In the **Monitoring (classic)** section of your Storage account's menu blade in the Azure portal, you can configure [Alert rules](#), such as sending email alerts to notify you when a specific metric reaches a certain value.

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. You must download the minute metrics for analysis. The tables do not 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 (see [Azure Storage Client Tools](#) for a list of available tools).

Metrics	Table names	Notes
Hourly metrics	\$MetricsHourPrimaryTransactionsBlob \$MetricsHourPrimaryTransactionsTable \$MetricsHourPrimaryTransactionsQueue \$MetricsHourPrimaryTransactionsFile	In versions prior to 2013-08-15, these tables were known as:  \$MetricsTransactionsBlob  \$MetricsTransactionsTable  \$MetricsTransactionsQueue  Metrics for the File service are available beginning with version 2015-04-05.
Minute metrics	\$MetricsMinutePrimaryTransactionsBlob \$MetricsMinutePrimaryTransactionsTable \$MetricsMinutePrimaryTransactionsQueue \$MetricsMinutePrimaryTransactionsFile	Can only be enabled using PowerShell or programmatically.  Metrics for the File service are available beginning with version 2015-04-05.
Capacity	\$MetricsCapacityBlob	Blob service only.

You can find full details of the schemas for these tables at [Storage Analytics Metrics Table Schema](#). The sample rows below show only a subset of the columns available, but illustrate some important features of the way Storage Metrics saves these metrics:

PartitionKey	RowKey	Timestamp	TotalRequests	TotalBillionRequests	TotalIngress	TotalEgress	Availability	AverageE2ELatency	AverageServerLatency	PercentSuccess
20140522T1100	user;All	2014-05-22T11:01:16.650250Z	7	7	4003	46801	100	104.4286	6.857143	100

20140 522T1 100	user;Q ueryEn tities	2014- 05- 22T11: 01:16.7 64025 0Z	5	5	2694	45951	100	143.8	7.8	100
20140 522T1 100	user;Q ueryEn tity	2014- 05- 22T11: 01:16.7 65025 0Z	1	1	538	633	100	3	3	100
20140 522T1 100	user;U pdateE ntity	2014- 05- 22T11: 01:16.7 65025 0Z	1	1	771	217	100	9	6	100

In this example minute metrics data, the partition key uses the time at minute resolution. The row key identifies the type of information that is stored in the row and this is composed of two pieces of information, 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 is a summary line, or it identifies the specific API such as **QueryEntity** or **UpdateEntity**.

The sample data above 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 add up to seven, which is the total shown on the **user>All** row. Similarly, you can derive the average end-to-end latency 104.4286 on the **userAll** row by calculating  $((143.8 * 5) + 3 + 9)/7$ .

## Metrics alerts

You should consider setting up alerts in the [Azure portal](#) so you will be automatically notified of important changes in the behavior of your storage services. 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. See [Azure Storage Client Tools](#) for a list of available storage explorer tools. You can configure alerts in the **Alert (classic)** blade, accessible under **Monitoring (classic)** in the Storage account menu blade.

### IMPORTANT

There may 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 may be written at once. This can lead to transactions from earlier minutes being aggregated into the transaction for the current minute. When this happens, the alert service may not have all available metrics data for the configured alert interval, which may lead to alerts firing unexpectedly.

## Accessing 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.

```
private static void PrintMinuteMetrics(CloudAnalyticsClient analyticsClient, DateTimeOffset startTime,
DateTimeOffset endTime)
{
 // Convert the dates to the format used in the PartitionKey
 var start = startTime.UnixTime.ToString("yyyyMMdd'T'HHmm");
 var end = endTime.UnixTime.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 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;
}
```

## 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 and delete requests of metrics data by a client are also billable at standard rates. If you have configured a data retention policy, you are not charged when Azure Storage deletes old metrics data. However, if you delete analytics data, your account is charged for the delete operations.

The capacity used by the metrics tables is also billable. You can use the following to estimate the amount of capacity used for storing metrics data:

- If each hour a service utilizes every API in every service, then approximately 148KB of data is stored every hour in the metrics transaction tables if you have enabled both service- and API-level summary.
- If within each hour, a service utilizes every API in the service, then approximately 12KB of data is stored every hour in the metrics transaction tables if you have enabled just service-level summary.
- The capacity table for blobs has two rows added each day, provided you have opted-in for logs. This implies

that every day, the size of this table increases by up to approximately 300 bytes.

## Next steps

- [How To Monitor a Storage Account](#)
- [Storage Analytics Metrics Table Schema](#)
- [Storage Analytics Logged Operations and Status Messages](#)
- [Storage Analytics Logging](#)

# Azure Storage analytics logging

4/3/2020 • 8 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.

Storage Analytics logging is not enabled by default for your storage account. You can enable it in 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 [Get Blob Service Properties](#), [Get Queue Service Properties](#), and [Get Table Service Properties](#) operations to enable Storage Analytics for each service.

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. However, premium storage account is not supported.

## 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.

## 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-AzureStorageBlob -Container '$logs' |
Where-Object {
 $_.Name -match 'table/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>

ATTRIBUTE	DESCRIPTION
MM	The two digit month for the log. For example: 07
DD	The two digit day for the log. For example: 31
hh	The two digit hour that indicates the starting hour for the logs, in 24 hour UTC format. For example: 18
mm	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 00.
<counter>	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 000000. For example: 000001

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
LogType	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: write  Example 2: read,write  Example 3: read,write,delete
StartTime	The earliest time of an entry in the log, in the form of YYYY-MM-DDThh:mm:ssZ. For example: 2011-07-31T18:21:46Z
EndTime	The latest time of an entry in the log, in the form of YYYY-MM-DDThh:mm:ssZ. For example: 2011-07-31T18:22:09Z
LogVersion	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

## Enable Storage logging

You can enable Storage logging with Azure portal, PowerShell, and Storage SDKs.

### Enable Storage logging using the Azure portal

In the Azure portal, use the **Diagnostics settings (classic)** blade to control Storage Logging, accessible from the **Monitoring (classic)** section of a storage account's **Menu blade**.

You can specify the storage services that you want to log, and the retention period (in days) for the logged data.

### Enable Storage logging using PowerShell

You can use PowerShell on your local machine to configure Storage Logging in your storage account by using the Azure PowerShell cmdlet **Get-AzureStorageServiceLoggingProperty** to retrieve the current settings, and the cmdlet **Set-AzureStorageServiceLoggingProperty** to change the current settings.

The cmdlets that control Storage Logging use a **LoggingOperations** parameter that is a string containing a comma-separated list of request types to log. The three possible request types are **read**, **write**, and **delete**. To switch off logging, use the value **none** for the **LoggingOperations** parameter.

The following command switches on logging for read, write, and delete requests in the Queue service in your default storage account with retention set to five days:

```
Set-AzureStorageServiceLoggingProperty -ServiceType Queue -LoggingOperations read,write,delete -RetentionDays 5
```

The following command switches off logging for the table service in your default storage account:

```
Set-AzureStorageServiceLoggingProperty -ServiceType Table -LoggingOperations none
```

For information about how to configure the Azure PowerShell cmdlets to work with your Azure subscription and how to select the default storage account to use, see: [How to install and configure Azure PowerShell](#).

### Enable Storage logging programmatically

In addition to using the Azure portal or the Azure PowerShell cmdlets to control Storage Logging, you can also use one of the Azure Storage APIs. For example, if you are using a .NET language you can use the Storage Client Library.

The classes **CloudBlobClient**, **CloudQueueClient**, and **CloudTableClient** all have methods such as **SetServiceProperties** and **SetServicePropertiesAsync** that take a **ServiceProperties** object as a parameter. You can use the **ServiceProperties** object to configure Storage Logging. For example, the following C# snippet shows how to change what is logged and the retention period for queue logging:

```
var storageAccount = CloudStorageAccount.Parse(connStr);
var queueClient = storageAccount.CreateCloudQueueClient();
var serviceProperties = queueClient.GetServiceProperties();

serviceProperties.Logging.LoggingOperations = LoggingOperations.All;
serviceProperties.Logging.RetentionDays = 2;

queueClient.SetServiceProperties(serviceProperties);
```

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](#).

## Download Storage logging 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 specific files](#).

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, including Microsoft Message Analyzer (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

- [Storage Analytics Log Format](#)
- [Storage Analytics Logged Operations and Status Messages](#)
- [Storage Analytics Metrics \(classic\)](#)

# Reacting to Blob storage events

4/7/2020 • 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:

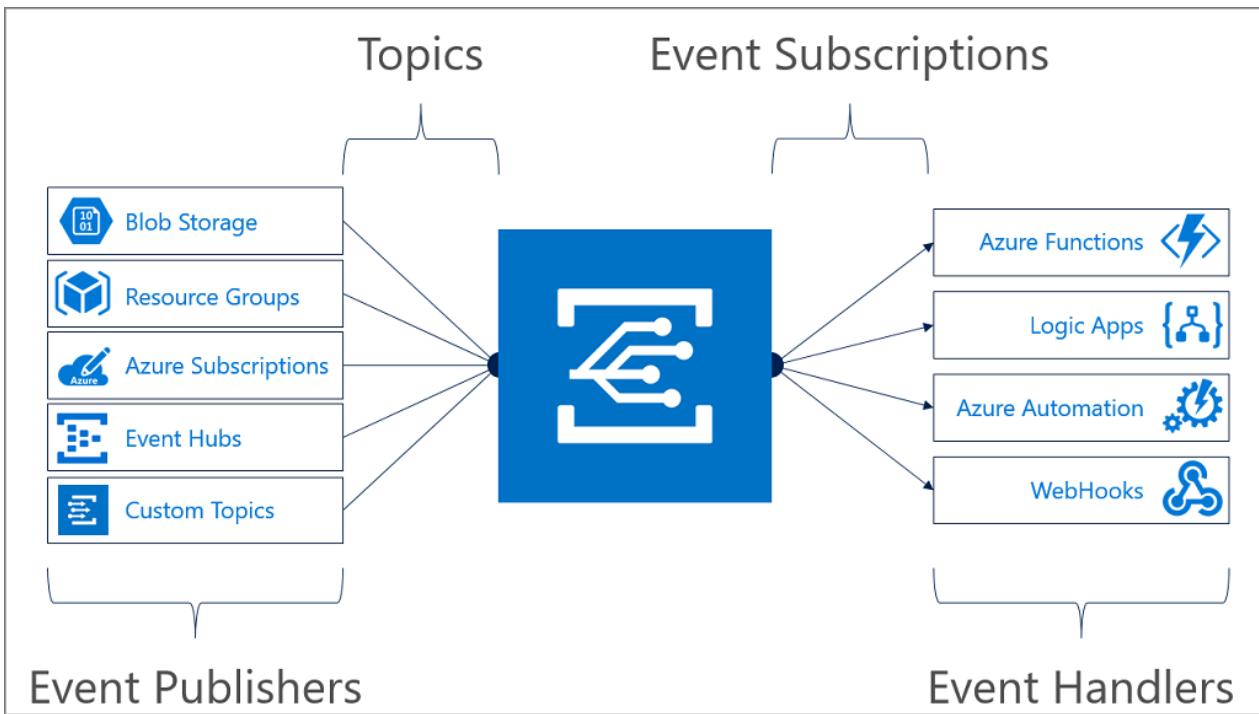
- [Tutorial: Use Azure Data Lake Storage Gen2 events to update a Databricks Delta table](#).
- [Tutorial: Automate resizing uploaded images using Event Grid](#)

## NOTE

Only storage accounts of kind **StorageV2** (general purpose v2), **BlockBlobStorage**, and **BlobStorage** support event integration. **Storage** (genral 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 or availability of subscriptions, duplicate messages may occasionally 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](#).

## 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](#)

# Change feed support in Azure Blob Storage (Preview)

3/18/2020 • 12 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.

## NOTE

This feature 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](#).

The change feed is 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.

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.

## 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 and is 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 GPv2 and Blob storage accounts can enable Change feed. Premium BlockBlobStorage accounts, and hierarchical namespace enabled accounts are not currently supported. GPv1 storage accounts are not supported but can be upgraded to GPv2 with no downtime, see [Upgrade to a GPv2 storage account](#) for more information.

## IMPORTANT

The change feed is in public preview, and is available in the **westcentralus** and **westus2** regions. See the [conditions](#) section of this article. To enroll in the preview, see the [Register your subscription](#) section of this article. You must register your subscription before you can enable change feed on your storage accounts.

- [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 **Blob Service**.
3. Click **Enabled** under **Blob change feed**
4. Choose the **Save** button to confirm your Data Protection settings

The screenshot shows the 'Data protection' section of the Azure Storage Blob service settings. It includes options for 'Blob soft delete' (Enabled), 'Retention policies' (set to retain for 7 days), and 'Blob change feed' (Enabled). The left sidebar lists various storage settings like Access keys, Geo-replication, and Configuration, with 'Data Protection' currently selected.

## 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 `$blobchangefeed` container is not visible in Azure Storage Explorer or the Azure portal. You currently cannot see the `$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](#).

## Understand change feed organization

### 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 interest to you.

Name	Blob Type	Blob Tier	Length
Content Type			
\$blobchangefeed/idx/segments/1601/01/01/0000/meta.json application/json	BlockBlob		584
\$blobchangefeed/idx/segments/2019/02/22/1810/meta.json application/json	BlockBlob		584
\$blobchangefeed/idx/segments/2019/02/22/1910/meta.json application/json	BlockBlob		584
\$blobchangefeed/idx/segments/2019/02/23/0110/meta.json application/json	BlockBlob		584

#### 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`).

Here's an example of change event record from change feed file converted to Json.

```
{
 "schemaVersion": 1,
 "topic": "/subscriptions/dd40261b-437d-43d0-86cf-
ef222b78fd15/resourceGroups/sadodd/providers/Microsoft.Storage/storageAccounts/mytestaccount",
 "subject": "/blobServices/default/containers/mytestcontainer/blobs/mytestblob",
 "eventType": "BlobCreated",
 "eventTime": "2019-02-22T18:12:01.079Z",
 "id": "55e5531f-8006-0000-00da-ca3467000000",
 "data": {
 "api": "PutBlob",
 "clientRequestId": "edf598f4-e501-4750-a3ba-9752bb22df39",
 "requestId": "00000000-0000-0000-0000-000000000000",
 "etag": "0x8D698F13DCB47F6",
 "contentType": "application/octet-stream",
 "contentLength": 128,
 "blobType": "BlockBlob",
 "url": "",
 "sequencer": "000000000000000100000000000000060000000000006d8a",
 "storageDiagnostics": {
 "bid": "11cda41c-13d8-49c9-b7b6-bc55c41b3e75",
 "seq": "(6,5614,28042,28038)",
 "sid": "591651bd-8eb3-c864-1001-fcd187be3efd"
 }
 }
}
```

For a description of each property, see [Azure Event Grid event schema for Blob Storage](#).

#### 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.

## 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"
 }
 }
}
```

## Register your subscription (Preview)

Because the change feed is only in public preview, you'll need to register your subscription to use the feature.

### Register by using PowerShell

In a PowerShell console, run these commands:

```
Register-AzProviderFeature -FeatureName Changefeed -ProviderNamespace Microsoft.Storage
Register-AzResourceProvider -ProviderNamespace Microsoft.Storage
```

### Register by using Azure CLI

In Azure Cloud Shell, run these commands:

```
az feature register --namespace Microsoft.Storage --name Changefeed
az provider register --namespace 'Microsoft.Storage'
```

## Conditions and known issues (Preview)

This section describes known issues and conditions in the current public preview of the change feed.

- For preview, you must first [register your subscription](#) before you can enable change feed for your storage account in the westcentralus or westus2 regions.
- The change feed captures only create, update, delete, and copy operations. Metadata updates are not currently captured in preview.
- Change event records for any single change might appear more than once in your change feed.

- You can't yet manage the lifetime of change feed log files by setting time-based retention policy on them and you cannot delete the blobs
- 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
- 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 during preview.

## FAQ

### **What is the difference between 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.

Change feed is a solution that provides transactional log of successful mutations or changes to your account such as blob creation, modification, and deletions. 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. Change feed is fundamentally designed and optimized for application development that require certain guarantees.

### **Should I use Change feed or Storage events?**

You can leverage both features as 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. 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](#)

# Overview of Azure page blobs

8/13/2019 • 9 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.

## 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:

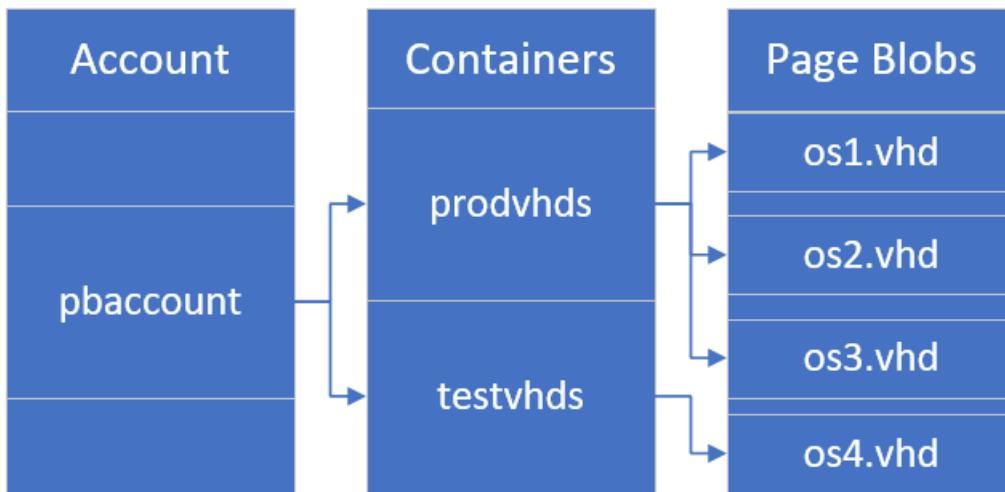
- Application-directed incremental snapshot management: Applications can leverage page blob snapshots and REST APIs for saving the application checkpoints without incurring costly duplication of data. Azure Storage supports local snapshots for page blobs, which don't require copying the entire blob. 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.

# 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

To create a page blob, we first create a `CloudBlobClient` object, with the base URI for accessing the blob storage for your storage account (`pbaccount` in figure 1) along with the `StorageCredentialsAccountAndKey` object, as shown in the following example. The example then shows creating a reference to a `CloudBlobContainer` object, and then creating the container (`testvhds`) if it doesn't already exist. Then using the `CloudBlobContainer` object, create a reference to a `CloudPageBlob` object by specifying the page blob name (`os4.vhd`) to access. To create the page blob, call `CloudPageBlob.Create`, passing in the max size for the blob to create. The `blobSize` must be a multiple of 512 bytes.

```
using Microsoft.Azure;
using Microsoft.Azure.Storage;
using Microsoft.Azure.Storage.Blob;

long OneGigabyteAsBytes = 1024 * 1024 * 1024;
// Retrieve storage account from connection string.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
 CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create the blob client.
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

// Retrieve a reference to a container.
CloudBlobContainer container = blobClient.GetContainerReference("testvhds");

// Create the container if it doesn't already exist.
container.CreateIfNotExists();

CloudPageBlob pageBlob = container.GetPageBlobReference("os4.vhd");
pageBlob.Create(16 * OneGigabyteAsBytes);
```

### Resizing a page blob

To resize a page blob after creation, use the `Resize` method. The requested size should be a multiple of 512 bytes.

```
pageBlob.Resize(32 * OneGigabyteAsBytes);
```

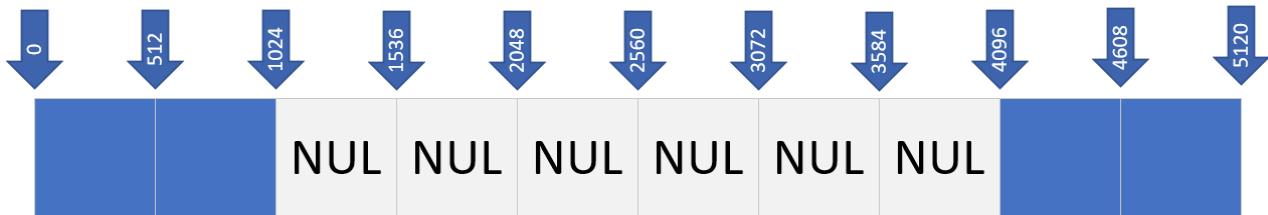
## Writing pages to a page blob

To write pages, use the [CloudPageBlob.WritePages](#) method. 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 ( $\text{startingOffset} \% 512 == 0$ ), and end on a 512 boundary - 1. The following code example shows how to call **WritePages** for a blob:

```
pageBlob.WritePages(dataStream, startingOffset);
```

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:



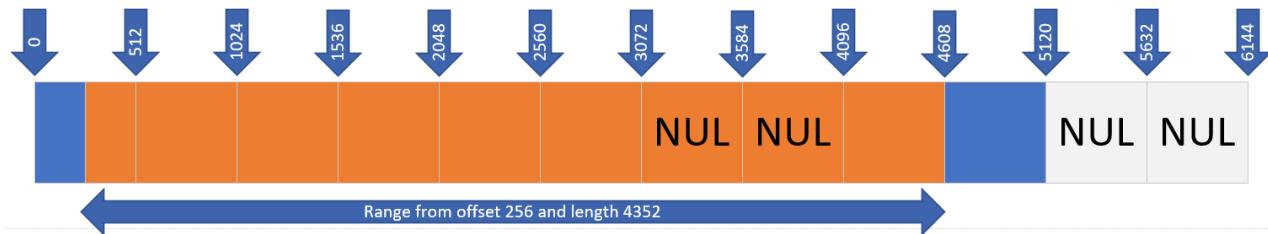
1. A Write operation starting at offset 0 of length 1024 bytes
2. A Write operation starting at offset 4096 of length 1024

## Reading pages from a page blob

To read pages, use the [CloudPageBlob.DownloadRangeToByteArray](#) method to read a range of bytes from the page blob. 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.

```
byte[] buffer = new byte[rangeSize];
pageBlob.DownloadRangeToByteArray(buffer, bufferOffset, pageBlobOffset, rangeSize);
```

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. To determine which pages are backed by data, use [CloudPageBlob.GetPageRanges](#). You can then enumerate the returned ranges and download the data in each range.

```

IEnumerable<PageRange> pageRanges = pageBlob.GetPageRanges();

foreach (PageRange range in pageRanges)
{
 // Calculate the range size
 int rangeSize = (int)(range.EndOffset + 1 - range.StartOffset);

 byte[] buffer = new byte[rangeSize];

 // Read from the correct starting offset in the page blob and
 // place the data in the bufferOffset of the buffer byte array
 pageBlob.DownloadRangeToByteArray(buffer, bufferOffset, range.StartOffset, rangeSize);

 // Then use the buffer for the page range just read
}

```

### **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

3/13/2020 • 4 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.

## NOTE

This feature 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](#).

## NOTE

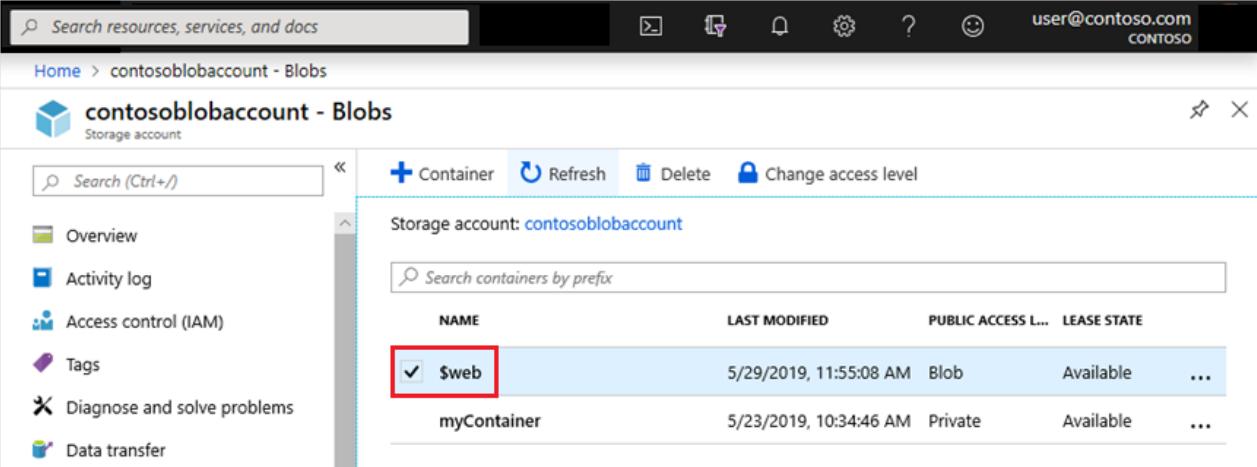
If your site depends on server-side code, use [Azure App Service](#) instead.

## 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 Blob container list for the 'contosoblobaccount' storage account. The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, and Data transfer. The main area displays a table of containers. A red box highlights the '\$web' row, which is checked under the 'Container' column. The table columns are NAME, LAST MODIFIED, PUBLIC ACCESS L..., and LEASE STATE. The '\$web' container was last modified on 5/29/2019 at 11:55:08 AM, is a Blob, and has Public access set to Available. The 'myContainer' row is also listed, last modified on 5/23/2019 at 10:34:46 AM, is a Private container, and has Public access set to Available.

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](#)

## 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. Use this table as a guide.

TOOL	GUIDANCE
Azure portal	<a href="#">Find the website URL by using the Azure portal</a>
Azure CLI	<a href="#">Find the website URL by using the Azure CLI</a>
Azure PowerShell module	<a href="#">Find the website URL by using PowerShell</a>

The URL of your site contains a regional code. For example the URL

`https://contosoblobaccount.z22.web.core.windows.net/` contains regional code `z22`.

While that code must remain in the URL, it is only for internal use, and you won't have to use that code in any other way.

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

CORS is not supported with static website.

## 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:

The screenshot shows the Azure Storage Blobs blade for a storage account named 'storagesamples'. On the left, there's a navigation menu with links like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Data transfer, Events, Storage Explorer (preview), Settings (Access keys, Geo-replication, CORS, Configuration), and Container (which is selected). In the main area, there's a search bar and a toolbar with Container, Refresh, Delete, and Change access level buttons. A red box highlights the 'Change access level' button. Below it is a 'Change access level' dialog with a description: 'Change the access level of all selected containers.' Under 'Public access level', there are four options: Private (no anonymous access) (selected), Private (no anonymous access), Blob (anonymous read access for blobs only), and Container (anonymous read access for containers and blobs). A list of containers is shown below, with sample-container4 and sample-container5 checked.

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](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.

## 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](#).

## 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](#).

## 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](#)

# Create an Azure Storage account

2/28/2020 • 8 minutes to read • [Edit Online](#)

An Azure storage account contains all of your Azure Storage data objects: blobs, files, queues, tables, and disks. The storage account provides a unique namespace for your Azure Storage data that is accessible from anywhere in the world over HTTP or HTTPS. Data in your Azure storage account is durable and highly available, secure, and massively scalable.

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](#).

## NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

## Prerequisites

If you don't have an Azure subscription, create a [free account](#) before you begin.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [Template](#)

None.

## Sign in to Azure

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [Template](#)

Sign in to the [Azure portal](#).

## Create a storage account

Now you are ready to create a storage account.

Every 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 article shows how to create a new resource group.

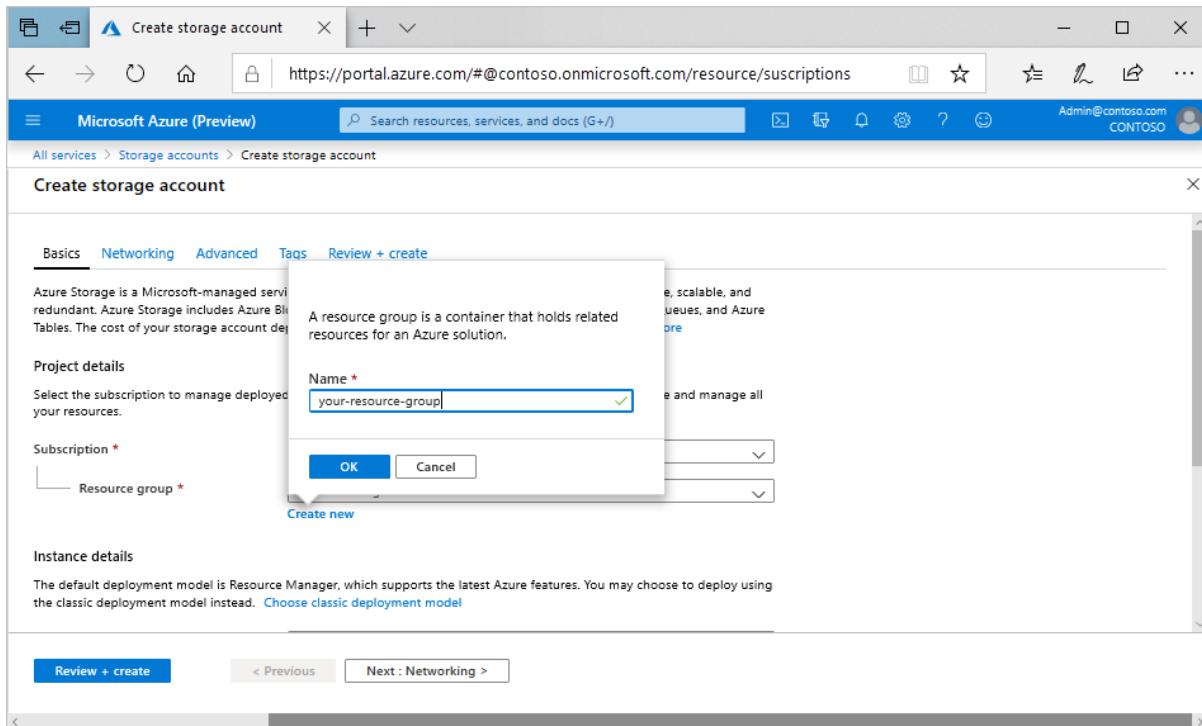
A **general-purpose v2** storage account provides access to all of the Azure Storage services: blobs, files, queues, tables, and disks. The steps outlined here create a general-purpose v2 storage account, but the steps to create any type of storage account are similar.

- [Portal](#)

- [PowerShell](#)
- [Azure CLI](#)
- [Template](#)

To create a general-purpose v2 storage account in the Azure portal, follow these steps:

1. On the Azure portal menu, select **All services**. In the list of resources, type **Storage Accounts**. As you begin typing, the list filters based on your input. Select **Storage Accounts**.
2. On the **Storage Accounts** window that appears, choose **Add**.
3. Select the subscription in which to create the storage account.
4. Under the **Resource group** field, select **Create new**. Enter a name for your new resource group, as shown in the following image.



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 can include numbers and lowercase letters only.
6. Select a location for your storage account, or use the default location.
7. Leave these fields set to their default values:

FIELD	VALUE
Deployment model	Resource Manager
Performance	Standard
Account kind	StorageV2 (general-purpose v2)
Replication	Read-access geo-redundant storage (RA-GRS)
Access tier	Hot

8. If you plan to use [Azure Data Lake Storage](#), choose the **Advanced** tab, and then set **Hierarchical namespace to Enabled**.

9. Select **Review + Create** to review your storage account settings and create the account.
  10. Select **Create**.
- For more information about types of storage accounts and other storage account settings, see [Azure storage account overview](#). For more information on resource groups, see [Azure Resource Manager overview](#).
- For more information about available replication options, see [Storage replication options](#).

## Delete a storage account

Deleting a storage account deletes the entire account, including all data in the account, and cannot be undone.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [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](#).

### WARNING

It's not possible to restore a deleted storage account or retrieve any of the content that it contained before deletion. Be sure to back up anything you want to save before you delete the account. This also holds true for any resources in the account—once you delete a blob, table, queue, or file, it is permanently deleted.

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](#).

## Next steps

In this how-to article, you've created a general-purpose v2 standard storage account. To learn how to upload and download blobs to and from your storage account, continue to one of the Blob storage quickstarts.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [Template](#)

[Work with blobs using the Azure portal](#)

# Create a BlockBlobStorage account

3/18/2020 • 4 minutes to read • [Edit Online](#)

The BlockBlobStorage account kind lets you create block blobs with premium performance characteristics. This type of storage account is optimized for workloads with high transaction rates or that require very fast access times. This article shows how to create a BlockBlobStorage account by using the Azure portal, the Azure CLI, or Azure PowerShell.

## NOTE

This feature 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](#).

For more information about BlockBlobStorage accounts, see [Azure storage account overview](#).

## Prerequisites

If you don't have an Azure subscription, create a [free account](#) before you begin.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

None.

## Sign in to Azure

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

Sign in to the [Azure portal](#).

## Create a BlockBlobStorage account

- [Portal](#)
- [Azure Powershell](#)
- [Azure CLI](#)

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

1. In the Azure portal, select **All services** > the **Storage** category > **Storage accounts**.
2. Under **Storage accounts**, select **Add**.
3. In the **Subscription** field, select the subscription in which to create the storage account.
4. In the **Resource group** field, select an existing resource group or select **Create new**, and enter a name for the new resource group.
5. In the **Storage account name** field, enter a name for the account. Note the following guidelines:

FIELD	VALUE
Performance	Select Premium.
Account kind	Select BlockBlobStorage.
Replication	Leave the default setting of Locally-redundant storage (LRS).

The screenshot shows the Azure portal's 'Create storage account' interface. On the left, there's a sidebar with various service links like Home, Dashboard, All services, and Favorites. The main area has a title 'Create storage account' and tabs for 'Basics', 'Advanced', 'Tags', and 'Review + create'. Under 'Basics', there's a brief introduction about Azure Storage. The 'PROJECT DETAILS' section allows selecting a subscription (Visual Studio Ultimate with MSDN) and a resource group ((New) resourcegroup1). The 'INSTANCE DETAILS' section is where the storage account is being configured. It includes fields for the storage account name (exampleaccountname), location (West US), performance level (Premium selected), account kind (BlockBlobStorage selected), and replication type (Locally-redundant storage (LRS) selected). There are also informational notes about each setting.

8. Select **Review + create** to review the storage account settings.

9. Select **Create**.

## Next steps

- For more information about storage accounts, see [Azure storage account overview](#).
- For more information about resource groups, see [Azure Resource Manager overview](#).

# Upgrade to a general-purpose v2 storage account

3/20/2020 • 10 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.

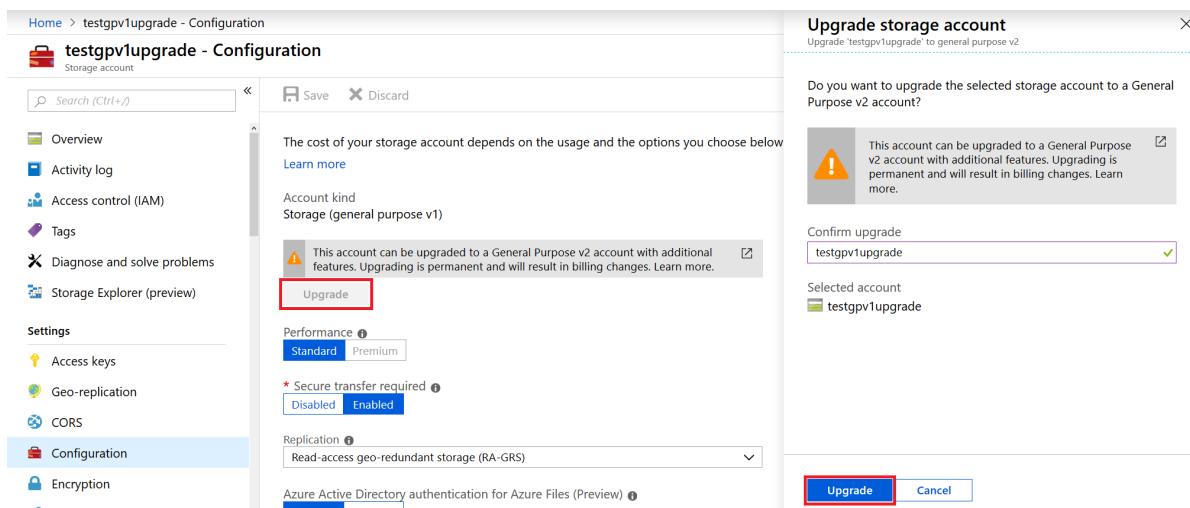
Upgrading to a general-purpose v2 storage account from your general-purpose v1 or Blob storage accounts is straightforward. You can upgrade using the Azure portal, PowerShell, or Azure CLI. There is no downtime or risk of data loss associated with upgrading to a general-purpose v2 storage account. The 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.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

1. Sign in to the [Azure portal](#).
2. Navigate to your storage account.
3. In the **Settings** section, click **Configuration**.
4. Under **Account kind**, click on **Upgrade**.
5. Under **Confirm upgrade**, type in the name of your account.
6. Click **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 apply 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

- [Create a storage account](#)

# Get storage account type and SKU name with .NET

3/12/2020 • 2 minutes to read • [Edit Online](#)

This article shows how to get the Azure Storage account type and SKU name for a blob by using the [Azure Storage client library for .NET](#).

Account information is available on service versions beginning with version 2018-03-28.

## About account type and SKU name

**Account type:** Valid account types include `BlobStorage`, `BlockBlobStorage`, `FileStorage`, `Storage`, and `StorageV2`. [Azure storage account overview](#) has more information, including descriptions of the various storage accounts.

**SKU name:** Valid SKU names include `Premium_LRS`, `Premium_ZRS`, `Standard_GRS`, `Standard_GZRS`, `Standard_LRS`, `Standard_RAGRS`, `Standard_RAGZRS`, and `Standard_ZRS`. SKU names are case-sensitive and are string fields in the [SkuName Class](#).

## Retrieve account information

To get the storage account type and SKU name associated with a blob, call the [GetAccountProperties](#) or [GetAccountPropertiesAsync](#) method.

The following code example retrieves and displays the read-only account properties.

```
private static async Task GetAccountInfoAsync(CloudBlob blob)
{
 try
 {
 // Get the blob's storage account properties.
 AccountProperties acctProps = await blob.GetAccountPropertiesAsync();

 // Display the properties.
 Console.WriteLine("Account properties");
 Console.WriteLine(" AccountKind: {0}", acctProps.AccountKind);
 Console.WriteLine(" SkuName: {0}", acctProps.SkuName);
 }
 catch (StorageException e)
 {
 Console.WriteLine("HTTP error code {0}: {1}",
 e.RequestInformation.HttpStatusCode,
 e.RequestInformation.ErrorCode);
 Console.WriteLine(e.Message);
 Console.ReadLine();
 }
}
```

## 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](#)

## **Next steps**

Learn about other operations you can perform on a storage account through the [Azure portal](#) and the Azure REST API.

- [Get Account Information operation \(REST\)](#)

# Create or delete a container in Azure Storage with .NET

3/12/2020 • 4 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 and delete 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 are not 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:

- [Create](#)
- [CreateAsync](#)
- [CreateIfNotExists](#)
- [CreateIfNotExistsAsync](#)

The `Create` and `CreateAsync` methods throw an exception if a container with the same name already exists.

The `CreateIfNotExists` and `CreateIfNotExistsAsync` methods return a Boolean value indicating whether the container was created. If a container with the same name already exists, then these methods return `False` to indicate that a new container was not created.

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:

```
private static async Task<CloudBlobContainer> CreateSampleContainerAsync(CloudBlobClient blobClient)
{
 // Name the sample container based on new GUID, to ensure uniqueness.
 // The container name must be lowercase.
 string containerName = "container-" + Guid.NewGuid();

 // Get a reference to a sample container.
 CloudBlobContainer container = blobClient.GetContainerReference(containerName);

 try
 {
 // Create the container if it does not already exist.
 bool result = await container.CreateIfNotExistsAsync();
 if (result == true)
 {
 Console.WriteLine("Created container {0}", container.Name);
 }
 }
 catch (StorageException e)
 {
 Console.WriteLine("HTTP error code {0}: {1}",
 e.RequestInformation.HttpStatusCode,
 e.RequestInformation.ErrorCode);
 Console.WriteLine(e.Message);
 }

 return container;
}
```

## 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`. You must explicitly create or delete the root container.

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 resides in the root container in the following manner:

```
https://myaccount.blob.core.windows.net/default.html
```

The following example creates the root container synchronously:

```
private static void CreateRootContainer(CloudBlobClient blobClient)
{
 try
 {
 // Create the root container if it does not already exist.
 CloudBlobContainer container = blobClient.GetContainerReference("$root");
 if (container.CreateIfNotExists())
 {
 Console.WriteLine("Created root container.");
 }
 else
 {
 Console.WriteLine("Root container already exists.");
 }
 }
 catch (StorageException e)
 {
 Console.WriteLine("HTTP error code {0}: {1}",
 e.RequestInformation.HttpStatusCode,
 e.RequestInformation.ErrorCode);
 Console.WriteLine(e.Message);
 }
}
```

## 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 does not exist.

The **DeleteIfExists** and **DeleteIfExistsAsync** methods return a Boolean value indicating whether the container was deleted. If the specified container does not exist, then these methods return **False** to indicate that the container was not deleted.

After you delete a container, you cannot create a container with the same name for at least 30 seconds, and possibly longer. While the container is being deleted, an attempt 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) while the container is being deleted.

The following example deletes the specified container, and handles the exception if the container does not exist:

```

private static async Task DeleteSampleContainerAsync(CloudBlobClient blobClient, string containerName)
{
 CloudBlobContainer container = blobClient.GetContainerReference(containerName);

 try
 {
 // Delete the specified container and handle the exception.
 await container.DeleteAsync();
 }
 catch (StorageException e)
 {
 Console.WriteLine("HTTP error code {0}: {1}",
 e.RequestInformation.HttpStatusCode,
 e.RequestInformation.ErrorCode);
 Console.WriteLine(e.Message);
 Console.ReadLine();
 }
}

```

The following example shows how to delete all of the containers that start with a specified prefix. The example breaks the lease if there is an existing lease on the container.

```

private static async Task DeleteContainersWithPrefixAsync(CloudBlobClient blobClient, string prefix)
{
 Console.WriteLine("Delete all containers beginning with the specified prefix");
 try
 {
 foreach (var container in blobClient.ListContainers(prefix))
 {
 Console.WriteLine("\tContainer: " + container.Name);
 if (container.Properties.LeaseState == LeaseState.Leased)
 {
 await container.BreakLeaseAsync(null);
 }

 await container.DeleteAsync();
 }

 Console.WriteLine();
 }
 catch (StorageException e)
 {
 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**

- [Create Container operation](#)
- [Delete Container operation](#)

# List blob containers with .NET

3/12/2020 • 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 .NET](#).

## Understand container listing options

To list containers in your storage account, call one of the following methods:

- [ListContainersSegmented](#)
- [ListContainersSegmentedAsync](#)

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 `maxresults` parameter when calling one of the [ListContainerSegmented](#) methods.

If your storage account contains more than 5000 containers, or if you have specified a value for `maxresults` 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 null. When the continuation token is null, then the set of results is complete. If the continuation token is not null, then call [ListContainersSegmented](#) or [ListContainersSegmentedAsync](#) again, passing in the continuation token to retrieve the next set of results, until the continuation token is null.

### 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 [ContainerListingDetails](#) enumeration. Azure Storage includes metadata with each container returned, so you do not need to also call one of the [FetchAttributes](#) methods to retrieve 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 in increments of 5 results at a time, and uses the continuation token to get the next segment of results. The example also returns container metadata with the results.

```

private static async Task ListContainersWithPrefixAsync(CloudBlobClient blobClient,
 string prefix)
{
 Console.WriteLine("List all containers beginning with prefix {0}, plus container metadata:", prefix);

 try
 {
 ContainerResultSegment resultSegment = null;
 BlobContinuationToken continuationToken = null;

 do
 {
 // List containers beginning with the specified prefix, returning segments of 5 results each.
 // Passing null for the maxResults parameter returns the max number of results (up to 5000).
 // Requesting the container's metadata with the listing operation populates the metadata,
 // so it's not necessary to also call FetchAttributes() to read the metadata.
 resultSegment = await blobClient.ListContainersSegmentedAsync(
 prefix, ContainerListingDetails.Metadata, 5, continuationToken, null, null);

 // Enumerate the containers returned.
 foreach (var container in resultSegment.Results)
 {
 Console.WriteLine("\tContainer: " + container.Name);

 // Write the container's metadata keys and values.
 foreach (var metadataItem in container.Metadata)
 {
 Console.WriteLine("\t\tMetadata key: " + metadataItem.Key);
 Console.WriteLine("\t\tMetadata value: " + metadataItem.Value);
 }
 }

 // Get the continuation token. If not null, get the next segment.
 continuationToken = resultSegment.ContinuationToken;
 } while (continuationToken != null);
 }
 catch (StorageException e)
 {
 Console.WriteLine("HTTP error code {0} : {1}",
 e.RequestInformation.HttpStatusCode,
 e.RequestInformation.ErrorCode);
 Console.WriteLine(e.Message);
 }
}
}

```

## 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

[List Containers](#) [Enumerating Blob Resources](#)

# Manage container properties and metadata with .NET

3/12/2020 • 3 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.

Retrieving property and metadata values for a Blob storage resource is a two-step process. Before you can read these values, you must explicitly fetch them by calling the `FetchAttributes` or `FetchAttributesAsync` method. The exception to this rule is that the `Exists` and `ExistsAsync` methods call the appropriate `FetchAttributes` method under the covers. When you call one of these methods, you do not need to also call `FetchAttributes`.

### IMPORTANT

If you find that property or metadata values for a storage resource have not been populated, then check that your code calls the `FetchAttributes` or `FetchAttributesAsync` method.

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:

- [FetchAttributes](#)
- [FetchAttributesAsync](#)

The following code example fetches a container's system properties and writes some property values to a console window:

```

private static async Task ReadContainerPropertiesAsync(CloudBlobContainer container)
{
 try
 {
 // Fetch some container properties and write out their values.
 await container.FetchAttributesAsync();
 Console.WriteLine("Properties for container {0}", container.StorageUri.PrimaryUri);
 Console.WriteLine("Public access level: {0}", container.Properties.PublicAccess);
 Console.WriteLine("Last modified time in UTC: {0}", container.Properties.LastModified);
 }
 catch (StorageException e)
 {
 Console.WriteLine("HTTP error code {0}: {1}",
 e.RequestInformation.HttpStatusCode,
 e.RequestInformation.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](#)

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 returns HTTP error code 400 (Bad Request).

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

```

public static async Task AddContainerMetadataAsync(CloudBlobContainer container)
{
 try
 {
 // Add some metadata to the container.
 container.Metadata.Add("docType", "textDocuments");
 container.Metadata["category"] = "guidance";

 // Set the container's metadata.
 await container.SetMetadataAsync();
 }
 catch (StorageException e)
 {
 Console.WriteLine("HTTP error code {0}: {1}",
 e.RequestInformation.HttpStatusCode,
 e.RequestInformation.ErrorCode);
 Console.WriteLine(e.Message);
 Console.ReadLine();
 }
}

```

To retrieve metadata, call the **FetchAttributes** or **FetchAttributesAsync** method on your blob or container to populate the **Metadata** collection, then read the values, as shown in the example below.

```

public static async Task ReadContainerMetadataAsync(CloudBlobContainer container)
{
 try
 {
 // Fetch container attributes in order to populate the container's properties and metadata.
 await container.FetchAttributes();

 // Enumerate the container's metadata.
 Console.WriteLine("Container metadata:");
 foreach (var metadataItem in container.Metadata)
 {
 Console.WriteLine("\tKey: {0}", metadataItem.Key);
 Console.WriteLine("\tValue: {0}", metadataItem.Value);
 }
 }
 catch (StorageException e)
 {
 Console.WriteLine("HTTP error code {0}: {1}",
 e.RequestInformation.HttpStatusCode,
 e.RequestInformation.ErrorCode);
 Console.WriteLine(e.Message);
 Console.ReadLine();
 }
}

```

## 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

- [Get Container Properties operation](#)
- [Set Container Metadata operation](#)
- [Get Container Metadata operation](#)

# List blobs with .NET

3/12/2020 • 5 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.

This article shows how to list blobs using the [Azure Storage client library for .NET](#).

## Understand blob listing options

To list the blobs in a storage account, call one of these methods:

- [CloudBlobClient.ListBlobs](#)
- [CloudBlobClient.ListBlobsSegmented](#)
- [CloudBlobClient.ListBlobsSegmentedAsync](#)

To list the blobs in a container, call one of these methods:

- [CloudBlobContainer.ListBlobs](#)
- [CloudBlobContainer.ListBlobsSegmented](#)
- [CloudBlobContainer.ListBlobsSegmentedAsync](#)

The overloads for these methods provide additional options for managing how blobs 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 `maxresults` parameter when calling one of the [ListBlobs](#) methods.

If a listing operation returns more than 5000 blobs, or if you have specified a value for `maxresults` 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 blobs. 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 null. When the continuation token is null, then the set of results is complete. If the continuation token is not null, then call listing operation again, passing in the continuation token to retrieve the next set of results, until the continuation token is null.

### 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 blobs whose names start with that prefix.

### Return metadata

To return blob metadata with the results, specify the **Metadata** value for the [BlobListingDetails](#) enumeration. Azure Storage includes metadata with each blob returned, so you do not need to call one of the [FetchAttributes](#) methods in this context to retrieve the blob metadata.

### 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.

```
private static async Task ListBlobsFlatListingAsync(CloudBlobContainer container, int? segmentSize)
{
 BlobContinuationToken continuationToken = null;
 CloudBlob blob;

 try
 {
 // Call the listing operation and enumerate the result segment.
 // When the continuation token is null, the last segment has been returned
 // and execution can exit the loop.
 do
 {
 BlobResultSegment resultSegment = await container.ListBlobsSegmentedAsync(string.Empty,
 true, BlobListingDetails.Metadata, segmentSize, continuationToken, null, null);

 foreach (var blobItem in resultSegment.Results)
 {
 // A flat listing operation returns only blobs, not virtual directories.
 blob = (CloudBlob)blobItem;

 // Write out some blob properties.
 Console.WriteLine("Blob name: {0}", blob.Name);
 }

 Console.WriteLine();

 // Get the continuation token and loop until it is null.
 continuationToken = resultSegment.ContinuationToken;
 } while (continuationToken != null);
 }
 catch (StorageException 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. The [Prefix](#) property of each virtual directory is set so that you can pass the prefix in a recursive call to retrieve the next directory.

To list blobs hierarchically, set the `useFlatBlobListing` parameter of the listing method to **false**.

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 the console window.

```

private static async Task ListBlobsHierarchicalListingAsync(CloudBlobContainer container, string prefix)
{
 CloudBlobDirectory dir;
 CloudBlob blob;
 BlobContinuationToken continuationToken = null;

 try
 {
 // Call the listing operation and enumerate the result segment.
 // When the continuation token is null, the last segment has been returned and
 // execution can exit the loop.
 do
 {
 BlobResultSegment resultSegment = await container.ListBlobsSegmentedAsync(prefix,
 false, BlobListingDetails.Metadata, null, continuationToken, null, null);
 foreach (var blobItem in resultSegment.Results)
 {
 // A hierarchical listing may return both virtual directories and blobs.
 if (blobItem is CloudBlobDirectory)
 {
 dir = (CloudBlobDirectory)blobItem;

 // Write out the prefix of the virtual directory.
 Console.WriteLine("Virtual directory prefix: {0}", dir.Prefix);

 // Call recursively with the prefix to traverse the virtual directory.
 await ListBlobsHierarchicalListingAsync(container, dir.Prefix);
 }
 else
 {
 // Write out the name of the blob.
 blob = (CloudBlob)blobItem;

 Console.WriteLine("Blob name: {0}", blob.Name);
 }
 Console.WriteLine();
 }

 // Get the continuation token and loop until it is null.
 continuationToken = resultSegment.ContinuationToken;

 } while (continuationToken != null);
 }
 catch (StorageException 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.

## 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](#)

## Next steps

- [List Blobs](#)
- [Enumerating Blob Resources](#)

# Manage blob properties and metadata with .NET

3/12/2020 • 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 .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.

Retrieving metadata and property values for a Blob storage resource is a two-step process. Before you can read these values, you must explicitly fetch them by calling the `FetchAttributes` or `FetchAttributesAsync` method. The exception to this rule is that the `Exists` and `ExistsAsync` methods call the appropriate `FetchAttributes` method under the covers. When you call one of these methods, you don't need to also call `FetchAttributes`.

### IMPORTANT

If you find that property or metadata values for a storage resource have not been populated, check that your code calls the `FetchAttributes` or `FetchAttributesAsync` method.

## Set and retrieve properties

The following code example sets the `ContentType` and `ContentLanguage` system properties on a blob.

```
public static async Task SetBlobPropertiesAsync(CloudBlob blob)
{
 try
 {
 Console.WriteLine("Setting blob properties.");

 // You must explicitly set the MIME ContentType every time
 // the properties are updated or the field will be cleared.
 blob.Properties.ContentType = "text/plain";
 blob.Properties.ContentLanguage = "en-us";

 // Set the blob's properties.
 await blob.SetPropertiesAsync();
 }
 catch (StorageException e)
 {
 Console.WriteLine("HTTP error code {0}: {1}",
 e.RequestInformation.HttpStatusCode,
 e.RequestInformation.ErrorCode);
 Console.WriteLine(e.Message);
 Console.ReadLine();
 }
}
```

To retrieve blob properties, call the `FetchAttributes` or `FetchAttributesAsync` method on your blob to populate the `Properties` property. The following code example gets a blob's system properties and displays some of the values:

```
private static async Task GetBlobPropertiesAsync(CloudBlob blob)
{
 try
 {
 // Fetch the blob properties.
 await blob.FetchAttributesAsync();

 // Display some of the blob's property values.
 Console.WriteLine(" ContentLanguage: {0}", blob.Properties.ContentLanguage);
 Console.WriteLine(" ContentType: {0}", blob.Properties.ContentType);
 Console.WriteLine(" Created: {0}", blob.Properties.Created);
 Console.WriteLine(" LastModified: {0}", blob.Properties.LastModified);
 }
 catch (StorageException e)
 {
 Console.WriteLine("HTTP error code {0}: {1}",
 e.RequestInformation.HttpStatusCode,
 e.RequestInformation.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(CloudBlob blob)
{
 try
 {
 // Add metadata to the blob by calling the Add method.
 blob.Metadata.Add("docType", "textDocuments");

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

 // Set the blob's metadata.
 await blob.SetMetadataAsync();
 }
 catch (StorageException e)
 {
 Console.WriteLine("HTTP error code {0}: {1}",
 e.RequestInformation.HttpStatusCode,
 e.RequestInformation.ErrorCode);
 Console.WriteLine(e.Message);
 Console.ReadLine();
 }
}

```

To retrieve metadata, call the `FetchAttributes` or `FetchAttributesAsync` method on your blob or container to populate the `Metadata` collection, then read the values, as shown in the example below.

```

public static async Task ReadBlobMetadataAsync(CloudBlob blob)
{
 try
 {
 // Fetch blob attributes in order to populate
 // the blob's properties and metadata.
 await blob.FetchAttributesAsync();

 Console.WriteLine("Blob metadata:");

 // Enumerate the blob's metadata.
 foreach (var metadataItem in blob.Metadata)
 {
 Console.WriteLine("\tKey: {0}", metadataItem.Key);
 Console.WriteLine("\tValue: {0}", metadataItem.Value);
 }
 }
 catch (StorageException e)
 {
 Console.WriteLine("HTTP error code {0}: {1}",
 e.RequestInformation.HttpStatusCode,
 e.RequestInformation.ErrorCode);
 Console.WriteLine(e.Message);
 Console.ReadLine();
 }
}

```

## 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**

- [Set Blob Properties operation](#)
- [Get Blob Properties operation](#)
- [Set Blob Metadata operation](#)
- [Get Blob Metadata operation](#)

# Copy a blob with .NET

3/12/2020 • 4 minutes to read • [Edit Online](#)

This article demonstrates how to copy a blob with an Azure Storage account. It also shows how to abort an asynchronous copy operation. The example code uses the [Azure Storage client library for .NET](#).

## About copying blobs

When you copy a blob within the same storage account, it is a synchronous operation. When you copy across accounts it is an asynchronous operation. The [StartCopy](#) and [StartCopyAsync](#) methods return a copy ID value that is used to check status or abort the copy 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. Any 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 blob 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.

For all blob types, you can check the [CopyState.Status](#) property on the destination blob to get the status of the copy operation. The final blob will be committed when the copy completes.

A copy operation can take any of the following forms:

- You can 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.
- You can 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.
- You can 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.
- You can 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.
- You can 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:

- [StartCopy](#)
- [StartCopyAsync](#)

The following code example gets a reference to a blob created previously and copies it to a new blob in the same container:

```

private static async Task CopyBlockBlobAsync(CloudBlobContainer container)
{
 CloudBlockBlob sourceBlob = null;
 CloudBlockBlob destBlob = null;
 string leaseId = null;

 try
 {
 // Get a block blob from the container to use as the source.
 sourceBlob = container.ListBlobs().OfType<CloudBlockBlob>().FirstOrDefault();

 // Lease the source blob for the copy operation to prevent another client from modifying it.
 // Specifying null for the lease interval creates an infinite lease.
 leaseId = await sourceBlob.AcquireLeaseAsync(null);

 // Get a reference to a destination blob (in this case, a new blob).
 destBlob = container.GetBlockBlobReference("copy of " + sourceBlob.Name);

 // Ensure that the source blob exists.
 if (await sourceBlob.ExistsAsync())
 {
 // Get the ID of the copy operation.
 string copyId = await destBlob.StartCopyAsync(sourceBlob);

 // Fetch the destination blob's properties before checking the copy state.
 await destBlob.FetchAttributesAsync();

 Console.WriteLine("Status of copy operation: {0}", destBlob.CopyState.Status);
 Console.WriteLine("Completion time: {0}", destBlob.CopyState.CompletionTime);
 Console.WriteLine("Bytes copied: {0}", destBlob.CopyState.BytesCopied.ToString());
 Console.WriteLine("Total bytes: {0}", destBlob.CopyState.TotalBytes.ToString());
 Console.WriteLine();
 }
 }
 catch (StorageException e)
 {
 Console.WriteLine(e.Message);
 Console.ReadLine();
 throw;
 }
 finally
 {
 // Break the lease on the source blob.
 if (sourceBlob != null)
 {
 await sourceBlob.FetchAttributes();

 if (sourceBlob.Properties.LeaseState != LeaseState.Available)
 {
 await sourceBlob.BreakLeaseAsync(new TimeSpan(0));
 }
 }
 }
}

```

## Abort a blob copy operation

Aborting a copy operation results in a destination blob of zero length for block blobs, append blobs, and page blobs. However, the metadata for the destination blob will have the new values copied from the source blob or set explicitly in the [StartCopy](#) or [StartCopyAsync](#) call. To keep the original metadata from before the copy, make a snapshot of the destination blob before calling [StartCopy](#) or [StartCopyAsync](#).

When you abort an ongoing blob copy operation, the destination blob's [CopyState.Status](#) is set to [CopyStatus.Aborted](#).

The [AbortCopy](#) and [AbortCopyAsync](#) methods cancel an ongoing blob copy operation, and leave a destination blob with zero length and full metadata.

```
// Fetch the destination blob's properties before checking the copy state.
await destBlob.FetchAttributesAsync();

// Check the copy status. If it is still pending, abort the copy operation.
if (destBlob.CopyState.Status == CopyStatus.Pending)
{
 await destBlob.AbortCopyAsync(copyId);
 Console.WriteLine("Copy operation {0} has been aborted.", copyId);
}
```

## 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](#)

## Next steps

The following topics contain information about copying blobs and aborting ongoing copy operations by using the Azure REST APIs.

- [Copy Blob](#)
- [Abort Copy Blob](#)

# Create and manage a blob snapshot in .NET

3/31/2020 • 6 minutes to read • [Edit Online](#)

A snapshot is a read-only version of a blob that's taken at a point in time. Snapshots are useful for backing up blobs. This article shows how to create and manage blob snapshots using the [Azure Storage client library for .NET](#).

## About blob snapshots

### NOTE

This feature 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](#).

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 URI is

`http://storagesample.core.blob.windows.net/mydrives/myvhds`, the snapshot URI is similar to

`http://storagesample.core.blob.windows.net/mydrives/myvhds?snapshot=2011-03-09T01:42:34.936000Z`.

### 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, meaning that a snapshot cannot outlive its 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.

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.

## Create a snapshot

To create a snapshot of a block blob, use one of the following methods:

- [CreateSnapshot](#)
- [CreateSnapshotAsync](#)

The following code example shows how to create a snapshot. This example specifies additional metadata for the snapshot when it is created.

```

private static async Task CreateBlockBlobSnapshot(CloudBlobContainer container)
{
 // Create a new block blob in the container.
 CloudBlockBlob baseBlob = container.GetBlockBlobReference("sample-base-blob.txt");

 // Add blob metadata.
 baseBlob.Metadata.Add("ApproxBlobCreatedDate", DateTime.UtcNow.ToString());

 try
 {
 // Upload the blob to create it, with its metadata.
 await baseBlob.UploadTextAsync(string.Format("Base blob: {0}", baseBlob.Uri.ToString()));

 // 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.
 Dictionary<string, string> metadata = new Dictionary<string, string>();
 metadata.Add("ApproxSnapshotCreatedDate", DateTime.UtcNow.ToString());
 await baseBlob.CreateSnapshotAsync(metadata, null, null, null);
 }
 catch (StorageException 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.

To delete blob snapshots, use one of the following blob deletion 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 `blockBlob` is an object of type [CloudBlockBlob](#):

```
await blockBlob.DeleteIfExistsAsync(DeleteSnapshotsOption.IncludeSnapshots, null, null, null);
```

## Return the absolute URI to a snapshot

The following code example creates a snapshot and writes out the absolute URI for the primary location.

```

//Create the blob service client object.
const string ConnectionString = "DefaultEndpointsProtocol=https;AccountName=account-name;AccountKey=account-
key";

CloudStorageAccount storageAccount = CloudStorageAccount.Parse(ConnectionString);
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

//Get a reference to a container.
CloudBlobContainer container = blobClient.GetContainerReference("sample-container");
container.CreateIfNotExists();

//Get a reference to a blob.
CloudBlockBlob blob = container.GetBlockBlobReference("sampleblob.txt");
blob.UploadText("This is a blob.");

//Create a snapshot of the blob and write out its primary URI.
CloudBlockBlob blobSnapshot = blob.CreateSnapshot();
Console.WriteLine(blobSnapshot.SnapshotQualifiedStorageUri.PrimaryUri);

```

## Understand how snapshots accrue charges

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.

### Important billing considerations

The following list includes key points to consider when creating a snapshot.

- Your storage account incurs charges for unique blocks or pages, whether they are in the blob or in the snapshot. Your account does not incur additional charges for snapshots associated with a blob until you update the blob on which they are based. After you update the base blob, it diverges from its snapshots. When this happens, you are charged for the unique blocks or pages in each blob or snapshot.
- 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 any 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.
- Replacing a block blob by calling the [UploadFromFile](#), [UploadText](#), [UploadFromStream](#), or [UploadFromByteArray](#) method replaces all blocks in the blob. If you have a snapshot associated with that blob, all blocks in the base blob and snapshot now diverge, and you will be charged for all the blocks in both blobs. This is true even if the data in the base blob and the snapshot remain identical.
- The Azure Blob service 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 consider that updating a blob that has a snapshot results in additional unique blocks and additional charges.

### Minimize cost with snapshot management

We recommend 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:

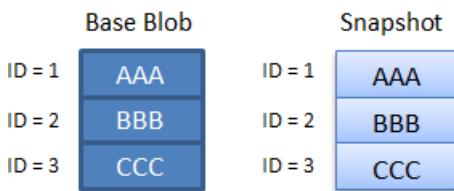
- Delete and re-create snapshots associated with a blob whenever you update the blob, even if you are updating with identical data, unless your application design requires that you maintain snapshots. By deleting and re-creating the blob's snapshots, you can ensure that the blob and snapshots do not diverge.
- If you are maintaining snapshots for a blob, avoid calling [UploadFromFile](#), [UploadText](#), [UploadFromStream](#), or [UploadFromByteArray](#) to update the blob. These methods replace all of the blocks in the blob, causing your base blob and its snapshots to diverge significantly. Instead, update the fewest possible number of blocks by using the [PutBlock](#) and [PutBlockList](#) methods.

## Snapshot billing scenarios

The following scenarios demonstrate how charges accrue for a block blob and its snapshots.

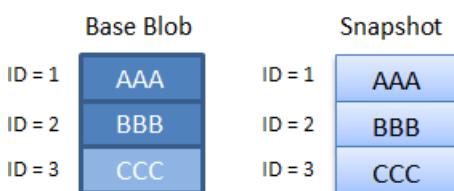
### Scenario 1

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.



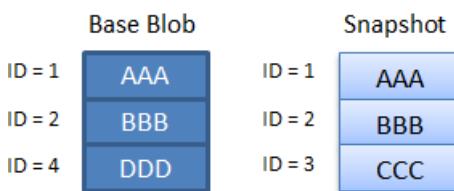
### Scenario 2

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.



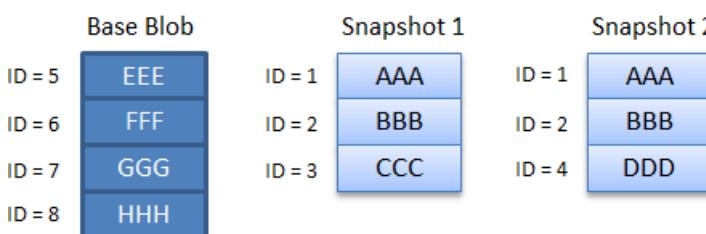
### Scenario 3

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.



### 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. This scenario can occur if you are using an update method such as [UploadFromFile](#), [UploadText](#), [UploadFromStream](#), or [UploadFromByteArray](#), because these methods replace all of the contents of a blob.



## Next steps

- You can find more information about working with virtual machine (VM) disk snapshots in [Back up Azure unmanaged VM disks with incremental snapshots](#)
- For additional code examples using Blob storage, see [Azure Code Samples](#). You can download a sample application and run it, or browse the code on GitHub.



# Use the Azure portal to access blob or queue data

4/14/2020 • 6 minutes to read • [Edit Online](#)

When you access blob or queue 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 or queue data

Depending on how you want to authorize access to blob or queue data in the Azure portal, you'll need specific permissions. In most cases, these permissions are provided via role-based access control (RBAC). For more information about RBAC, see [What is role-based access control \(RBAC\)?](#).

### Use the account access key

To access blob and queue data with the account access key, you must have an RBAC role assigned to you that includes the RBAC action `Microsoft.Storage/storageAccounts/listkeys/action`. This RBAC role may be a built-in or a custom role. Built-in roles that support `Microsoft.Storage/storageAccounts/listkeys/action` include:

- The Azure Resource Manager [Owner](#) role
- The Azure Resource Manager [Contributor](#) role
- The [Storage Account Contributor](#) role

When you attempt to access blob or queue 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 and queue data. If you have not been assigned a role with this action, then the portal attempts to access data using 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 and queue data with the account key. For more information, see [Classic subscription administrator roles, Azure RBAC roles, and Azure AD administrator roles](#).

### Use your Azure AD account

To access blob or queue data from the Azure portal using your Azure AD account, both of the following statements must be true for you:

- 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.
- You have been assigned either a built-in or custom role that provides access to blob or queue data.

The [Reader](#) role assignment or another Azure Resource Manager role assignment is necessary so that the user can view and navigate storage account management resources in the Azure portal. The RBAC roles that grant access to

blob or queue data do not grant access to storage account management resources. To access blob or queue data in the portal, the user needs permissions to navigate storage account resources. For more information about this requirement, see [Assign the Reader role for portal access](#).

The built-in roles that support access to your blob or queue data include:

- [Storage Blob Data Owner](#): For POSIX access control for Azure Data Lake Storage Gen2.
- [Storage Blob Data Contributor](#): Read/write/delete permissions for blobs.
- [Storage Blob Data Reader](#): Read-only permissions for blobs.
- [Storage Queue Data Contributor](#): Read/write/delete permissions for queues.
- [Storage Queue Data Reader](#): Read-only permissions for queues.

Custom roles can support different combinations of the same permissions provided by the built-in roles. For more information about creating custom RBAC roles, see [Custom roles for Azure resources](#) and [Understand role definitions for Azure resources](#).

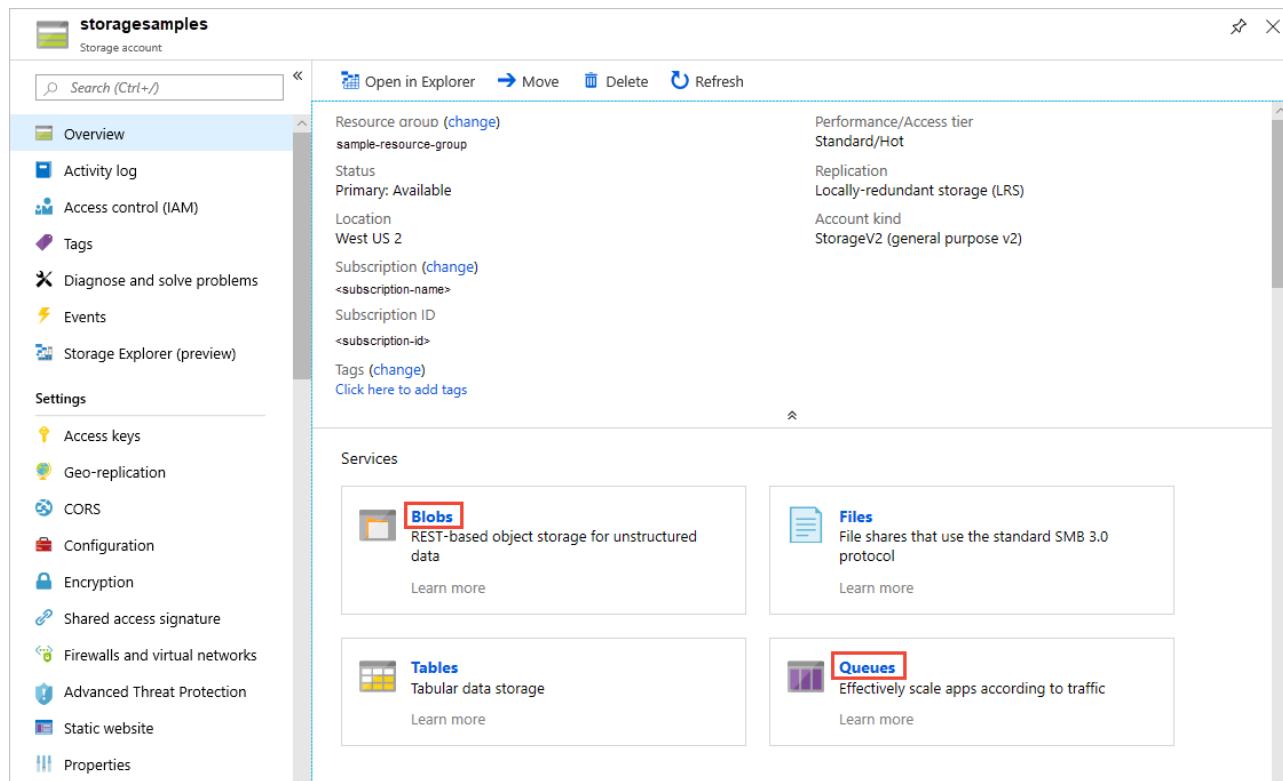
Listing queues with a classic subscription administrator role is not supported. To list queues, a user must have assigned to them the Azure Resource Manager Reader role, the **Storage Queue Data Reader** role, or the **Storage Queue Data Contributor** role.

#### IMPORTANT

The preview version of Storage Explorer in the Azure portal does not support using Azure AD credentials to view and modify blob or queue data. Storage Explorer in the Azure portal always uses the account keys to access data. To use Storage Explorer in the Azure portal, you must be assigned a role that includes `Microsoft.Storage/storageAccounts/listkeys/action`.

## Navigate to blobs or queues in the portal

To view blob or queue data in the portal, navigate to the **Overview** for your storage account, and click on the links for **Blobs** or **Queues**. Alternatively you can navigate to the **Blob service** and **Queue service** sections in the menu.



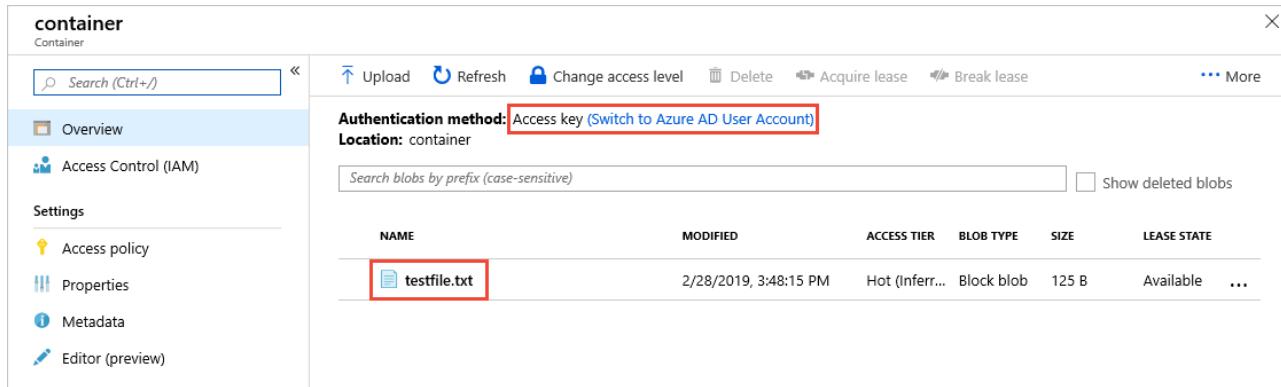
# Determine the current authentication method

When you navigate to a container or a queue, the Azure portal indicates whether you are currently using the account access key or your Azure AD account to authenticate.

The examples in this section show accessing a container and its blobs, but the portal displays the same message when you are accessing a queue and its messages, or listing queues.

## Authenticate with the account access key

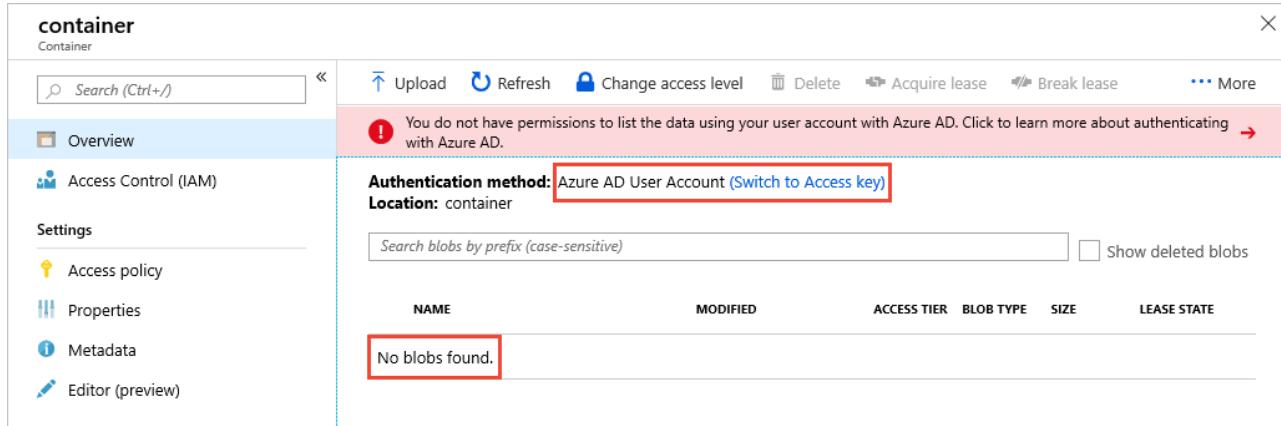
If you are authenticating using the account access key, you'll see **Access Key** specified as the authentication method in the portal:



The screenshot shows the Azure portal interface for a container named "container". On the left, there's a sidebar with options like Overview, Access Control (IAM), Settings, and Editor (preview). The main area shows blob details. At the top, it says "Authentication method: Access key (Switch to Azure AD User Account)". Below that, it says "Location: container". A search bar and a "Show deleted blobs" checkbox are present. A table lists blobs, with one entry for "testfile.txt" highlighted by a red box. The table columns are NAME, MODIFIED, ACCESS TIER, BLOB TYPE, SIZE, and LEASE STATE.

NAME	MODIFIED	ACCESS TIER	BLOB TYPE	SIZE	LEASE STATE
testfile.txt	2/28/2019, 3:48:15 PM	Hot (Infer...)	Block blob	125 B	Available

To switch to using Azure AD account, click the link highlighted in the image. If you have the appropriate permissions via the RBAC 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:



This screenshot shows the same Azure portal interface as the previous one, but with a different authentication method. It says "Authentication method: Azure AD User Account (Switch to Access key)" and "Location: container". A red box highlights this text. A message at the top 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." A red box also highlights this message. The blob list table below shows "No blobs found." A red box highlights this text.

NAME	MODIFIED	ACCESS TIER	BLOB TYPE	SIZE	LEASE STATE
No blobs found.					

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 Azure Storage container overview page for a container named 'container'. On the left, there's a navigation menu with options like Overview, Access Control (IAM), Settings (Access policy, Properties, Metadata, Editor (preview)), and a search bar. The main area displays blob details. At the top, it says 'Authentication method: Azure AD User Account (Switch to Access key)' and 'Location: container'. Below that is a search bar for blobs by prefix. A table lists blobs with columns: NAME, MODIFIED, ACCESS TIER, BLOB TYPE, SIZE, and LEASE STATE. One blob, 'testfile.txt', is highlighted with a red box around its name.

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 same Azure Storage container overview page as before, but with a different outcome. It now displays an error message: '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 listed as 'Access key (Switch to Azure AD User Account)'. The blob table shows 'No blobs found.', with a red box highlighting this text.

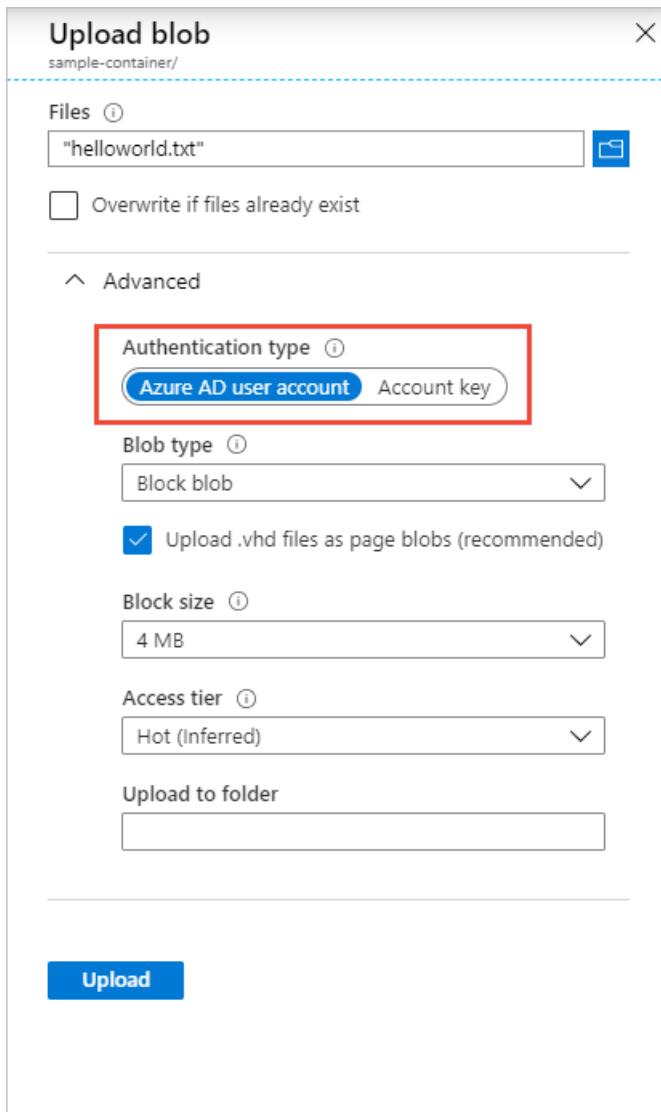
Notice that no blobs appear in the list if you do not have access to the account keys. Click on the [Switch to Azure AD User Account](#) link to use your Azure AD account for authentication again.

## 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:



## Next steps

- Authenticate access to Azure blobs and queues using Azure Active Directory
- Grant access to Azure containers and queues with RBAC in the Azure portal
- Grant access to Azure blob and queue data with RBAC using Azure CLI
- Grant access to Azure blob and queue data with RBAC using PowerShell

# Run PowerShell commands with Azure AD credentials to access blob or queue data

12/30/2019 • 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 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 and queue data to an Azure AD security principal via role-based access control (RBAC). For more information about RBAC roles in Azure Storage, see [Manage access rights to Azure Storage data with RBAC](#).

## Supported operations

The Azure Storage extensions are supported for operations on blob and queue 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 or queues are assigned via 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 or queue. If you have been assigned the **Blob Data Contributor** role, then you can run scripting commands that read, write, or delete a container or queue or the data they contain.

For details about the permissions required for each Azure Storage operation on a container or queue, see [Call storage operations with OAuth tokens](#).

## Call PowerShell commands using Azure AD credentials

### NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

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 RBAC roles, see [Grant access to Azure blob and queue data with RBAC in the Azure portal](#).

#### IMPORTANT

RBAC 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

- [Use PowerShell to assign an RBAC role for access to blob and queue data](#)
- [Authorize access to blob and queue data with managed identities for Azure resources](#)

# Authorize access to blob or queue data with Azure CLI

3/1/2020 • 5 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 or queue 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 and queue 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.

### 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 and queue data to an Azure AD security principal via role-based access control (RBAC). For more information about RBAC roles in Azure Storage, see [Manage access rights to Azure Storage data with RBAC](#).

### Permissions for calling data operations

The Azure Storage extensions are supported for operations on blob and queue 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 or queues are assigned via RBAC. For example, if you are assigned the

**Blob Data Reader** role, then you can run scripting commands that read data from a container or queue. If you are assigned the **Blob Data Contributor** role, then you can run scripting commands that read, write, or delete a container or queue or the data they contain.

For details about the permissions required for each Azure Storage operation on a container or queue, 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 log 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 RBAC roles, see [Grant access to Azure blob and queue data with RBAC in the Azure portal](#).

##### IMPORTANT

RBAC 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
```

## 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

- [Use Azure CLI to assign an RBAC role for access to blob and queue data](#)
- [Authorize access to blob and queue data with managed identities for Azure resources](#)

# Use the Azure portal to assign an RBAC role for access to blob and queue data

4/1/2020 • 8 minutes to read • [Edit Online](#)

Azure Active Directory (Azure AD) authorizes access rights to secured resources through [role-based access control \(RBAC\)](#). Azure Storage defines a set of built-in RBAC roles that encompass common sets of permissions used to access blob or queue data.

When an RBAC role is assigned to an Azure AD security principal, Azure grants access to those resources for that security principal. Access can be scoped to the level of the subscription, the resource group, the storage account, or an individual container or queue. An Azure AD security principal may be a user, a group, an application service principal, or a [managed identity for Azure resources](#).

This article describes how to use the Azure portal to assign RBAC roles. The Azure portal provides a simple interface for assigning RBAC roles and managing access to your storage resources. You can also assign RBAC roles for blob and queue resources using Azure command-line tools or the Azure Storage management APIs. For more information about RBAC roles for storage resources, see [Authenticate access to Azure blobs and queues using Azure Active Directory](#).

## RBAC roles for blobs and queues

Azure provides the following built-in RBAC roles for authorizing access to blob and queue data using Azure AD and OAuth:

- [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 Queue Data Contributor](#): Use to grant read/write/delete permissions to Azure queues.
- [Storage Queue Data Reader](#): Use to grant read-only permissions to Azure queues.
- [Storage Queue Data Message Processor](#): Use to grant peek, retrieve, and delete permissions to messages in Azure Storage queues.
- [Storage Queue Data Message Sender](#): Use to grant add permissions to messages in Azure Storage queues.

For detailed information about built-in RBAC 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 RBAC roles](#), and [Azure AD roles](#).

### NOTE

RBAC role assignments may take up to five minutes to propagate.

Only roles explicitly defined for data access permit a security principal to access blob or queue data. 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 or queue data within that account.

Access to blob or queue data in the Azure portal can be authorized using either your Azure AD account or the storage account access key. For more information, see [Use the Azure portal to access blob or queue data](#).

## Determine resource scope

Before you assign an 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.

The following list describes the levels at which you can scope access to Azure blob and queue resources, starting 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.
- **An individual queue.** At this scope, a role assignment applies to messages in the queue, as well as queue properties and metadata.
- **The storage account.** At this scope, a role assignment applies to all containers and their blobs, or to all queues and their messages.
- **The resource group.** At this scope, a role assignment applies to all of the containers or queues in all of the storage accounts in the resource group.
- **The subscription.** At this scope, a role assignment applies to all of the containers or queues in all of the storage accounts in all of the resource groups in the subscription.

### IMPORTANT

If your subscription includes an Azure DataBricks namespace, roles that are scoped to the subscription will not grant access to blob and queue data. Scope roles to the resource group, storage account, or container or queue instead.

## Assign RBAC roles using the Azure portal

After you have determined the appropriate scope for a role assignment, navigate to that resource in the Azure portal. Display the **Access Control (IAM)** settings for the resource, and follow these instructions to manage role assignments:

1. Assign the appropriate Azure Storage RBAC role to grant access to an Azure AD security principal.
2. Assign the Azure Resource Manager [Reader](#) role to users who need to access containers or queues via the Azure portal using their Azure AD credentials.

The following sections describe each of these steps in more detail.

### NOTE

As an owner of your Azure Storage account, you are not automatically assigned permissions to access data. You must explicitly assign yourself an RBAC role for Azure Storage. You can assign it at the level of your subscription, resource group, storage account, or a container or queue.

You cannot assign a role scoped to a container or queue if your storage account has a hierarchical namespace enabled.

## Assign a built-in RBAC role

Before you assign a role to a security principal, be sure to consider the scope of the permissions you are granting. Review the [Determine resource scope](#) section to decide the appropriate scope.

The procedure shown here assigns a role scoped to a container, but you can follow the same steps to assign a role scoped to a queue:

1. In the [Azure portal](#), go to your storage account and display the [Overview](#) for the account.
2. Under Services, select **Blobs**.

- Locate the container for which you want to assign a role, and display the container's settings.
- Select **Access control (IAM)** to display access control settings for the container. Select the **Role assignments** tab to see the list of role assignments.

The screenshot shows the Microsoft Azure portal interface for managing access control (IAM) in a storage account named 'sample-container'. The 'Access control (IAM)' tab is active. On the left, a sidebar lists various management options like Overview, Activity log, and Settings. The 'Access control (IAM)' link is highlighted with a red box. The main content area displays a table of role assignments:

Name	Type	Role	Scope
Contributor			
AD Admin	User	Contributor	Subscription (Inherited)
A2 Admin 2	User	Contributor	Subscription (Inherited)
RI Robert	User	Contributor	Subscription (Inherited)

- Click the **Add role assignment** button to add a new role.
- In the **Add role assignment** window, select the Azure Storage role that you want to assign. Then search to locate the security principal to which you want to assign that role.

The screenshot shows the 'Add role assignment' dialog box. It has three main sections: 'Role' (set to 'Storage Blob Data Reader'), 'Assign access to' (set to 'Azure AD user, group, or service principal'), and 'Select' (showing 'azure-user' selected). Below these is a 'Selected members:' list with one item: 'azure-user'. At the bottom are 'Save' and 'Discard' buttons.

- Click **Save**. The identity to whom you assigned the role appears listed under that role. For example, the following image shows that the user added now has read permissions to data in the container named *sample-container*.

The screenshot shows the Azure portal's 'Access control (IAM)' section for a storage account. It lists various roles and their assignments:

Role	Assignment
App	App2
Reader	Subscription (Inherited)
Storage Blob Data Reader (Preview)	AZ azuser azuser@ User Storage Blob Data Reader ... This resource

You can follow similar steps to assign a role scoped to the storage account, resource group, or subscription.

### Assign the Reader role for portal access

When you assign a built-in or custom role for Azure Storage to a security principal, you are granting permissions to that security principal to perform operations on data in your storage account. The built-in **Data Reader** roles provide read permissions for the data in a container or queue, while the built-in **Data Contributor** roles provide read, write, and delete permissions to a container or queue. Permissions are scoped to the specified resource. 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. Additional Azure AD permissions are required to navigate through the portal and view the other resources that are visible there.

If your users need to be able to access blobs in the Azure portal, then assign them an additional RBAC role, the **Reader** role, to those users, at the level of the storage account or above. 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.

Follow these steps to assign the **Reader** role so that a user can access blobs from the Azure portal. In this example, the assignment is scoped to the storage account:

1. In the [Azure portal](#), navigate to your storage account.
2. Select **Access control (IAM)** to display the access control settings for the storage account. Select the **Role assignments** tab to see the list of role assignments.
3. In the **Add role assignment** window, select the **Reader** role.
4. From the **Assign access to** field, select **Azure AD user, group, or service principal**.
5. Search to locate the security principal to which you want to assign the role.
6. Save the role assignment.

Assigning the **Reader** role is necessary only for users who need to access blobs or queues using the Azure portal.

#### IMPORTANT

The preview version of Storage Explorer in the Azure portal does not support using Azure AD credentials to view and modify blob or queue data. Storage Explorer in the Azure portal always uses the account keys to access data. To use Storage Explorer in the Azure portal, you must be assigned a role that includes **Microsoft.Storage/storageAccounts/listkeys/action**.

## Next steps

- For more information about RBAC roles for storage resources, see [Authenticate access to Azure blobs and queues using Azure Active Directory](#).

- To learn more about RBAC, see [What is role-based access control \(RBAC\)?](#).
- To learn how to assign and manage RBAC role assignments with Azure PowerShell, Azure CLI, or the REST API, see these articles:
  - [Manage role-based access control \(RBAC\) with Azure PowerShell](#)
  - [Manage role-based access control \(RBAC\) with Azure CLI](#)
  - [Manage role-based access control \(RBAC\) with the REST API](#)
- To learn how to authorize access to containers and queues from within your storage applications, see [Use Azure AD with Azure Storage applications](#).

# Use PowerShell to assign an RBAC role for access to blob and queue data

12/12/2019 • 6 minutes to read • [Edit Online](#)

Azure Active Directory (Azure AD) authorizes access rights to secured resources through [role-based access control \(RBAC\)](#). Azure Storage defines a set of built-in RBAC roles that encompass common sets of permissions used to access containers or queues.

When an RBAC role is assigned to an Azure AD security principal, Azure grants access to those resources for that security principal. Access can be scoped to the level of the subscription, the resource group, the storage account, or an individual container or queue. An Azure AD security principal may be a user, a group, an application service principal, or a [managed identity for Azure resources](#).

This article describes how to use Azure PowerShell to list built-in RBAC roles and assign them to users. For more information about using Azure PowerShell, see [Overview of Azure PowerShell](#).

## NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

## RBAC roles for blobs and queues

Azure provides the following built-in RBAC roles for authorizing access to blob and queue data using Azure AD and OAuth:

- [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 Queue Data Contributor](#): Use to grant read/write/delete permissions to Azure queues.
- [Storage Queue Data Reader](#): Use to grant read-only permissions to Azure queues.
- [Storage Queue Data Message Processor](#): Use to grant peek, retrieve, and delete permissions to messages in Azure Storage queues.
- [Storage Queue Data Message Sender](#): Use to grant add permissions to messages in Azure Storage queues.

For detailed information about built-in RBAC 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 RBAC roles](#), and [Azure AD roles](#).

#### NOTE

RBAC role assignments may take up to five minutes to propagate.

Only roles explicitly defined for data access permit a security principal to access blob or queue data. 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 or queue data within that account.

Access to blob or queue data in the Azure portal can be authorized using either your Azure AD account or the storage account access key. For more information, see [Use the Azure portal to access blob or queue data](#).

## Determine resource scope

Before you assign an 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.

The following list describes the levels at which you can scope access to Azure blob and queue resources, starting 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.
- **An individual queue.** At this scope, a role assignment applies to messages in the queue, as well as queue properties and metadata.
- **The storage account.** At this scope, a role assignment applies to all containers and their blobs, or to all queues and their messages.
- **The resource group.** At this scope, a role assignment applies to all of the containers or queues in all of the storage accounts in the resource group.
- **The subscription.** At this scope, a role assignment applies to all of the containers or queues in all of the storage accounts in all of the resource groups in the subscription.

#### IMPORTANT

If your subscription includes an Azure DataBricks namespace, roles that are scoped to the subscription will not grant access to blob and queue data. Scope roles to the resource group, storage account, or container or queue instead.

## List available RBAC roles

To list available built-in RBAC roles with Azure PowerShell, use the [Get-AzRoleDefinition](#) command:

```
Get-AzRoleDefinition | FT Name, Description
```

You'll see the built-in Azure Storage data roles listed, together with other built-in roles for Azure:

Storage Blob Data Contributor	Allows for read, write and delete access to Azure Storage blob containers and data
Storage Blob Data Owner	Allows for full access to Azure Storage blob containers and data, including assigning POSIX access control.
Storage Blob Data Reader	Allows for read access to Azure Storage blob containers and data
Storage Queue Data Contributor	Allows for read, write, and delete access to Azure Storage queues and queue messages
Storage Queue Data Message Processor	Allows for peek, receive, and delete access to Azure Storage queue messages
Storage Queue Data Message Sender	Allows for sending of Azure Storage queue messages
Storage Queue Data Reader	Allows for read access to Azure Storage queues and queue messages

# Assign an RBAC role to a security principal

To assign an RBAC role to a security principal, use the [New-AzRoleAssignment](#) command. The format of the command can differ based on the scope of the assignment. In order to run the command, you need to have Owner or Contributor role assigned at the corresponding scope. The following examples show how to assign a role to a user at various scopes, but you can use the same command to assign a role to any security principal.

## Container scope

To assign a role scoped to a container, specify a string containing the scope of the container for the `--scope` parameter. The scope for a container is in the form:

```
/subscriptions/<subscription>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>/blobServices/default/containers/<container-name>
```

The following example assigns the **Storage Blob Data Contributor** role to a user, scoped to a container named *sample-container*. Make sure to replace the sample values and the placeholder values in brackets with your own values:

```
New-AzRoleAssignment -SignInName <email> `
 -RoleDefinitionName "Storage Blob Data Contributor" `
 -Scope "/subscriptions/<subscription>/resourceGroups/sample-resource-group/providers/Microsoft.Storage/storageAccounts/<storage-account>/blobServices/default/containers/sample-container"
```

## Queue scope

To assign a role scoped to a queue, specify a string containing the scope of the queue for the `--scope` parameter. The scope for a queue is in the form:

```
/subscriptions/<subscription>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>/queueServices/default/queues/<queue-name>
```

The following example assigns the **Storage Queue Data Contributor** role to a user, scoped to a queue named *sample-queue*. Make sure to replace the sample values and the placeholder values in brackets with your own values:

```
New-AzRoleAssignment -SignInName <email> `
 -RoleDefinitionName "Storage Queue Data Contributor" `
 -Scope "/subscriptions/<subscription>/resourceGroups/sample-resource-group/providers/Microsoft.Storage/storageAccounts/<storage-account>/queueServices/default/queues/sample-queue"
```

## Storage account scope

To assign a role scoped to the storage account, specify the scope of the storage account resource for the `--scope` parameter. The scope for a storage account is in the form:

```
/subscriptions/<subscription>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>
```

The following example shows how to scope the **Storage Blob Data Reader** role to a user at the level of the storage account. Make sure to replace the sample values with your own values:

```
New-AzRoleAssignment -SignInName <email> `
 -RoleDefinitionName "Storage Blob Data Reader" `
 -Scope "/subscriptions/<subscription>/resourceGroups/sample-resource-
group/providers/Microsoft.Storage/storageAccounts/<storage-account>"
```

## Resource group scope

To assign a role scoped to the resource group, specify the resource group name or ID for the `--resource-group` parameter. The following example assigns the **Storage Queue Data Reader** role to a user at the level of the resource group. Make sure to replace the sample values and placeholder values in brackets with your own values:

```
New-AzRoleAssignment -SignInName <email> `
 -RoleDefinitionName "Storage Queue Data Reader" `
 -ResourceGroupName "sample-resource-group"
```

## Subscription scope

To assign a role scoped to the subscription, specify the scope for the subscription for the `--scope` parameter. The scope for a subscription is in the form:

```
/subscriptions/<subscription>
```

The following example shows how to assign the **Storage Blob Data Reader** role to a user at the level of the storage account. Make sure to replace the sample values with your own values:

```
New-AzRoleAssignment -SignInName <email> `
 -RoleDefinitionName "Storage Blob Data Reader" `
 -Scope "/subscriptions/<subscription>"
```

## Next steps

- [Manage access to Azure resources using RBAC and Azure PowerShell](#)
- [Grant access to Azure blob and queue data with RBAC using Azure CLI](#)
- [Grant access to Azure blob and queue data with RBAC in the Azure portal](#)

# Use Azure CLI to assign an RBAC role for access to blob and queue data

12/5/2019 • 6 minutes to read • [Edit Online](#)

Azure Active Directory (Azure AD) authorizes access rights to secured resources through [role-based access control \(RBAC\)](#). Azure Storage defines a set of built-in RBAC roles that encompass common sets of permissions used to access blob or queue data.

When an RBAC role is assigned to an Azure AD security principal, Azure grants access to those resources for that security principal. Access can be scoped to the level of the subscription, the resource group, the storage account, or an individual container or queue. An Azure AD security principal may be a user, a group, an application service principal, or a [managed identity for Azure resources](#).

This article describes how to use Azure CLI to list built-in RBAC roles and assign them to users. For more information about using Azure CLI, see [Azure Command-Line Interface \(CLI\)](#).

## RBAC roles for blobs and queues

Azure provides the following built-in RBAC roles for authorizing access to blob and queue data using Azure AD and OAuth:

- [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 Queue Data Contributor](#): Use to grant read/write/delete permissions to Azure queues.
- [Storage Queue Data Reader](#): Use to grant read-only permissions to Azure queues.
- [Storage Queue Data Message Processor](#): Use to grant peek, retrieve, and delete permissions to messages in Azure Storage queues.
- [Storage Queue Data Message Sender](#): Use to grant add permissions to messages in Azure Storage queues.

For detailed information about built-in RBAC 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 RBAC roles](#), and [Azure AD roles](#).

### NOTE

RBAC role assignments may take up to five minutes to propagate.

Only roles explicitly defined for data access permit a security principal to access blob or queue data. 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 or queue data within that account.

Access to blob or queue data in the Azure portal can be authorized using either your Azure AD account or the storage account access key. For more information, see [Use the Azure portal to access blob or queue data](#).

## Determine resource scope

Before you assign an 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.

The following list describes the levels at which you can scope access to Azure blob and queue resources, starting 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.
- **An individual queue.** At this scope, a role assignment applies to messages in the queue, as well as queue properties and metadata.
- **The storage account.** At this scope, a role assignment applies to all containers and their blobs, or to all queues and their messages.
- **The resource group.** At this scope, a role assignment applies to all of the containers or queues in all of the storage accounts in the resource group.
- **The subscription.** At this scope, a role assignment applies to all of the containers or queues in all of the storage accounts in all of the resource groups in the subscription.

#### IMPORTANT

If your subscription includes an Azure DataBricks namespace, roles that are scoped to the subscription will not grant access to blob and queue data. Scope roles to the resource group, storage account, or container or queue instead.

## List available RBAC roles

To list available built-in RBAC roles with Azure CLI, use the [az role definition list](#) command:

```
az role definition list --out table
```

You'll see the built-in Azure Storage data roles listed, together with other built-in roles for Azure:

Storage Blob Data Contributor	Allows for read, write and delete access to Azure Storage blob containers and data
Storage Blob Data Owner	Allows for full access to Azure Storage blob containers and data, including assigning POSIX access control.
Storage Blob Data Reader	Allows for read access to Azure Storage blob containers and data
Storage Queue Data Contributor	Allows for read, write, and delete access to Azure Storage queues and queue messages
Storage Queue Data Message Processor	Allows for peek, receive, and delete access to Azure Storage queue messages
Storage Queue Data Message Sender	Allows for sending of Azure Storage queue messages
Storage Queue Data Reader	Allows for read access to Azure Storage queues and queue messages

## Assign an RBAC role to a security principal

To assign an RBAC role to a security principal, use the [az role assignment create](#) command. The format of the command can differ based on the scope of the assignment. The following examples show how to assign a role to a user at various scopes, but you can use the same command to assign a role to any security principal.

### Container scope

To assign a role scoped to a container, specify a string containing the scope of the container for the `--scope` parameter. The scope for a container is in the form:

```
/subscriptions/<subscription>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>/blobServices/default/containers/<container>
```

The following example assigns the **Storage Blob Data Contributor** role to a user, scoped to the level of the container. Make sure to replace the sample values and the placeholder values in 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>/blobServices/default/containers/<container>"
```

## Queue scope

To assign a role scoped to a queue, specify a string containing the scope of the queue for the `--scope` parameter. The scope for a queue is in the form:

```
/subscriptions/<subscription>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>/queueServices/default/queues/<queue>
```

The following example assigns the **Storage Queue Data Contributor** role to a user, scoped to the level of the queue. Make sure to replace the sample values and the placeholder values in brackets with your own values:

```
az role assignment create \
--role "Storage Queue Data Contributor" \
--assignee <email> \
--scope "/subscriptions/<subscription>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>/queueServices/default/queues/<queue>"
```

## Storage account scope

To assign a role scoped to the storage account, specify the scope of the storage account resource for the `--scope` parameter. The scope for a storage account is in the form:

```
/subscriptions/<subscription>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>
```

The following example shows how to assign the **Storage Blob Data Reader** role to a user at the level of the storage account. Make sure to replace the sample values with your own values:

```
az role assignment create \
--role "Storage Blob Data Reader" \
--assignee <email> \
--scope "/subscriptions/<subscription>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>"
```

## Resource group scope

To assign a role scoped to the resource group, specify the resource group name or ID for the `--resource-group` parameter. The following example assigns the **Storage Queue Data Reader** role to a user at the level of the resource group. Make sure to replace the sample values and placeholder values in brackets with your own values:

```
az role assignment create \
--role "Storage Queue Data Reader" \
--assignee <email> \
--resource-group <resource-group>
```

## Subscription scope

To assign a role scoped to the subscription, specify the scope for the subscription for the `--scope` parameter. The scope for a subscription is in the form:

```
/subscriptions/<subscription>
```

The following example shows how to assign the **Storage Blob Data Reader** role to a user at the level of the storage account. Make sure to replace the sample values with your own values:

```
az role assignment create \
--role "Storage Blob Data Reader" \
--assignee <email> \
--scope "/subscriptions/<subscription>"
```

## Next steps

- [Manage access to Azure resources using RBAC and Azure PowerShell](#)
- [Grant access to Azure blob and queue data with RBAC using Azure PowerShell](#)
- [Grant access to Azure blob and queue data with RBAC in the Azure portal](#)

# Authorize access to blob and queue data with managed identities for Azure resources

1/14/2020 • 6 minutes to read • [Edit Online](#)

Azure Blob and Queue storage support Azure Active Directory (Azure AD) authentication with [managed identities for Azure resources](#). Managed identities for Azure resources can authorize access to blob and queue 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 or queue data from an Azure VM using managed identities for Azure Resources. It also describes how to test your code in the development environment.

## Enable managed identities on a VM

Before you can use managed identities for Azure Resources to authorize access to blobs and queues 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](#).

## Authenticate with the Azure Identity library

The Azure Identity client library provides Azure AD token authentication support for the [Azure SDK](#). The latest versions of the Azure Storage client libraries for .NET, Java, Python, and JavaScript integrate with the Azure Identity library to provide a simple and secure means to acquire an OAuth 2.0 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 authenticate whether your application is running in the development environment or in Azure. The Azure Identity client library for .NET authenticates a security principal. When your code is running in Azure, the security principal is a managed identity for Azure resources. In the development environment, the managed identity does not exist, so the client library authenticates either the user or a service principal for testing purposes.

After authenticating, the Azure Identity client library gets a token credential. This token credential is then encapsulated in the service client object that you create to perform operations against Azure Storage. The library handles this for you seamlessly by getting the appropriate token credential.

For more information about the Azure Identity client library for .NET, see [Azure Identity client library for .NET](#). For reference documentation for the Azure Identity client library, see [Azure.Identity Namespace](#).

## Assign role-based access control (RBAC) roles for access to data

When an Azure AD security principal attempts to access blob or queue 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 RBAC role that grants access to blob or queue data in Azure Storage. For information about assigning permissions via RBAC, see the section titled **Assign RBAC roles for access rights** in [Authorize access to Azure blobs and queues using Azure Active Directory](#).

## 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 supports 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](#).

Other development tools may prompt you to login via a web browser.

## 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.

### Create the service principal

To create a service principal with Azure CLI and assign an RBAC 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 [Built-in roles for Azure resources](#).

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 Reader** role to it with account scope

```
az ad sp create-for-rbac \
--name <service-principal> \
--role "Storage Blob Data Reader" \
--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"
}
```

### IMPORTANT

RBAC role assignments may take a few 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](#).

## 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
```

## .NET code example: Create a block blob

Add the following `using` directives to your code to use the Azure Identity and Azure Storage client libraries.

```
using Azure;
using Azure.Identity;
using Azure.Storage.Blobs;
using System;
using System.IO;
using System.Text;
using System.Threading.Tasks;
```

To get a token credential that your code can use to authorize requests to Azure Storage, create an instance of the [DefaultAzureCredential](#) class. The following code example shows how to get the authenticated token credential and use it to create a service client object, then use 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://'{0}'.blob.core.windows.net/{1}",
 accountName,
 containerName);

 // Get a credential and create a 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 or queue data with Azure AD, you must use HTTPS for those requests.

## Next steps

- [Manage access rights to storage data with RBAC.](#)
- [Use Azure AD with storage applications.](#)
- [Run Azure CLI or PowerShell commands with Azure AD credentials to access blob or queue data.](#)

# Acquire a token from Azure AD for authorizing requests from a client application

2/12/2020 • 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 (formerly Azure AD). 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 Microsoft identity platform 2.0. The code example features .NET, but other languages use a similar approach. For more information about Microsoft identity platform 2.0, see [Microsoft identity platform \(v2.0\) 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](#).

## Assign a role to an Azure AD security principal

To authenticate a security principal from your Azure Storage application, first configure role-based access control (RBAC) settings for that security principal. Azure Storage defines built-in RBAC roles that encompass permissions for containers and queues. When the RBAC role is assigned to a security principal, that security principal is granted access to that resource. For more information, see [Manage access rights to Azure Blob and Queue data with RBAC](#).

## 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:

## Register an application



**!** If you are building an application for external users that will be distributed by Microsoft, you must register as a first party application to meet all security, privacy, and compliance policies. [Read our decision guide](#)

### \* 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
- Accounts in any organizational directory
- Accounts in any organizational directory and personal Microsoft accounts (e.g. Skype, Xbox, Outlook.com)

[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.





[By proceeding, you agree to the Microsoft Platform Policies](#)

### 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**:

## StorageClientAppSample



<input type="text" value="Search (Ctrl+J)"/>	
	<a href="#">Overview</a>
	<a href="#">Quickstart</a>
	<a href="#">Manage</a>
	<a href="#">Branding</a>
	<a href="#">Authentication</a>

	<a href="#">Delete</a>		<a href="#">Endpoints</a>
Display name	StorageClientAppSample	Supported account types	<a href="#">My organization only</a>
Application (client) ID	<application-id>	Redirect URIs	1 web, 0 public client
Directory (tenant) ID	<directory-id>	Managed application in local directory	<a href="#">Create Service Principal</a>
Object ID	<object-id>		

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 **Overview** page for your registered application, select **View API Permissions**.
2. In the **API permissions** section, select **Add a permission** and choose **Microsoft APIs**.
3. Select **Azure Storage** from the list of results to display the **Request API permissions** pane.
4. 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.
5. In the **Select permissions** section of the **Request API permissions** pane, select the checkbox next to **user\_impersonation**, then click **Add permissions**.

The screenshot shows the 'Request API permissions' pane. At the top, there's a header with a back arrow labeled 'All APIs', the 'Azure Storage' logo, and a 'Docs' link. Below the header, a question asks 'What type of permissions does your application require?'. Two options are shown: 'Delegated permissions' (selected) and 'Application permissions'. The 'Delegated permissions' section includes a note: 'Your application needs to access the API as the signed-in user.' The 'Application permissions' section includes a note: 'Your application runs as a background service or daemon without a signed-in user.' Below this, a 'Select permissions' section has a search bar labeled 'Type to search'. A table lists permissions under 'PERMISSION' and 'ADMIN CONSENT REQUIRED'. One row is visible: 'user\_impersonation' (checkbox checked, 'Access Azure Storage' link), with 'ADMIN CONSENT REQUIRED' marked as '-'. At the bottom are 'Add permissions' and 'Discard' buttons.

The **API permissions** pane now shows that your registered Azure AD application has access to both Microsoft Graph and the Azure Storage. Permissions are granted to Microsoft Graph automatically when you first register your app with Azure AD.

The screenshot shows the 'API permissions' pane. At the top, it says 'API permissions' and provides a note: 'Applications are authorized to use APIs by requesting permissions. These permissions show up during the consent process where users are given the opportunity to grant/deny access.' Below is a table with columns: 'API / PERMISSIONS NAME', 'TYPE', 'DESCRIPTION', and 'ADMIN CONSENT REQUIRED'. The table shows two entries:

API / PERMISSIONS NAME	TYPE	DESCRIPTION	ADMIN CONSENT REQUIRED
▼ Azure Storage (1)			
user_impersonation	Delegated	Access Azure Storage	-
▼ Microsoft Graph (1)			
User.Read	Delegated	Sign in and read user profile	-

At the bottom, a note states: 'These are the permissions that this application requests statically. You may also request user consent-able permissions dynamically through code. [See best practices for requesting permissions](#)'.

## Create a client secret

The application needs a client secret to prove its identity when requesting a token. 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*****

## 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 content](#).

## 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. For instructions on how to assign this role in the Azure portal, see [Grant access to Azure blob and queue data with RBAC in the Azure portal](#).

#### 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 RBAC role for Azure Storage. You can assign it at the level of your subscription, resource group, storage account, or container or queue.

### 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:

```
Install-Package Microsoft.Azure.Storage.Blob
Install-Package Microsoft.Azure.Storage.Common
```

Next, add the following using statements to the `HomeController.cs` file:

```
using Microsoft.Identity.Client; //MSAL library for getting the access token
using Microsoft.WindowsAzure.Storage.Auth;
using Microsoft.WindowsAzure.Storage.Blob;
```

### Create a block blob

Add the following code snippet to create a block blob:

```
private static async Task<string> CreateBlob(string accessToken)
{
 // Create a blob on behalf of the user
 TokenCredential tokenCredential = new TokenCredential(accessToken);
 StorageCredentials storageCredentials = new StorageCredentials(tokenCredential);

 // Replace the URL below with your storage account URL
 CloudBlockBlob blob =
 new CloudBlockBlob(
 new Uri("https://<storage-account>.blob.core.windows.net/<container>/Blob1.txt"),
 storageCredentials);
 await blob.UploadTextAsync("Blob created by Azure AD authenticated user.");
 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 authorize the request 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 OAuth 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. When consent is necessary, the example catches the **MsalUiRequiredException** and calls another method to facilitate the request for consent:

```
public async Task<IActionResult> Blob()
{
 var scopes = new string[] { "https://storage.azure.com/user_impersonation" };
 try
 {
 var accessToken =
 await _tokenAcquisition.GetAccessTokenOnBehalfOfUser(HttpContext, scopes);
 ViewData["Message"] = await CreateBlob(accessToken);
 return View();
 }
 catch (MsalUiRequiredException ex)
 {
 AuthenticationProperties properties =
 BuildAuthenticationPropertiesForIncrementalConsent(scopes, ex);
 return Challenge(properties);
 }
}
```

Consent is the process of a user granting authorization to an application to access protected resources on their behalf. The Microsoft identity platform 2.0 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 at any given time by in the `scope` parameter. For more information about incremental consent, see the section titled **Incremental and dynamic consent** in [Why update to Microsoft identity platform \(v2.0\)?](#).

The following method constructs the authentication properties for requesting incremental consent:

```
private AuthenticationProperties BuildAuthenticationPropertiesForIncrementalConsent(string[] scopes,
 MsalUiRequiredException ex)
{
 AuthenticationProperties properties = new AuthenticationProperties();

 // Set the scopes, including the scopes that ADAL.NET or MSAL.NET need for the Token cache.
 string[] additionalBuildInScopes = new string[] { "openid", "offline_access", "profile" };
 properties.SetParameter<ICollection<string>>(OpenIdConnectParameterNames.Scope,
 scopes.Union(additionalBuildInScopes).ToList());

 // Attempt to set the login_hint so that the logged-in user is not presented
 // with an account selection dialog.
 string loginHint = HttpContext.User.GetLoginHint();
 if (!string.IsNullOrWhiteSpace(loginHint))
 {
 properties.SetParameter<string>(OpenIdConnectParameterNames.LoginHint, loginHint);

 string domainHint = HttpContext.User.GetDomainHint();
 properties.SetParameter<string>(OpenIdConnectParameterNames.DomainHint, domainHint);
 }

 // Specify any additional claims that are required (for instance, MFA).
 if (!string.IsNullOrEmpty(ex.Claims))
 {
 properties.Items.Add("claims", ex.Claims);
 }

 return properties;
}
```

[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

Next, 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>",
 "CallbackPath": "/signin-oidc",
 "SignedOutCallbackPath": "/signout-callback-oidc",

 // To call an API
 "ClientSecret": "<client-secret>"
 },
 "Logging": {
 "LogLevel": {
 "Default": "Warning"
 }
 },
 "AllowedHosts": "*"
}
```

## Update the storage account and container name

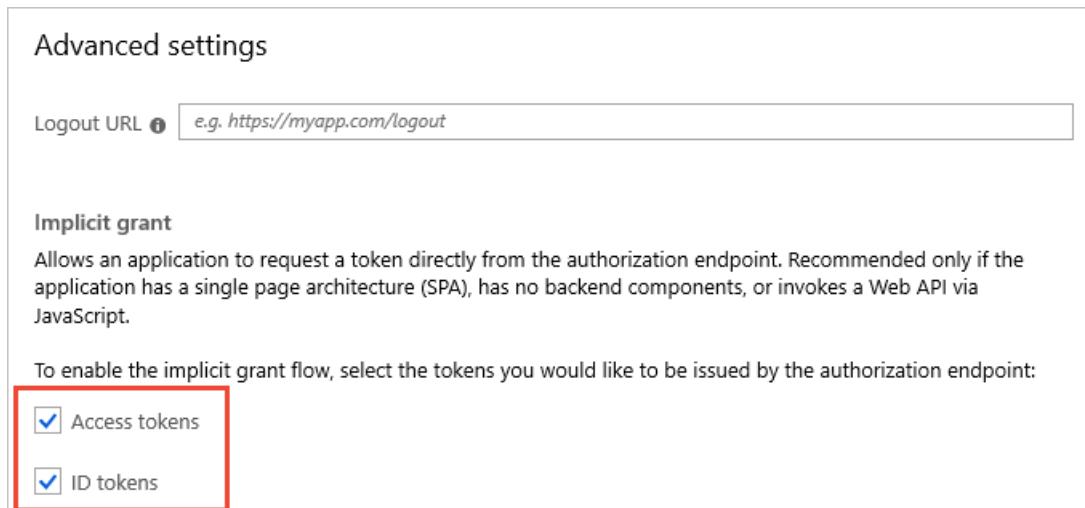
In the *HomeController.cs* file, update the URI that references the block blob to use the name of your storage account and container:

```
CloudBlockBlob blob = new CloudBlockBlob(
 new Uri("https://<storage-account>.blob.core.windows.net/<container>/Blob1.txt"),
 storageCredentials);
```

## Enable implicit grant flow

To run the sample, you may need to configure the implicit grant flow for your app registration. Follow these steps:

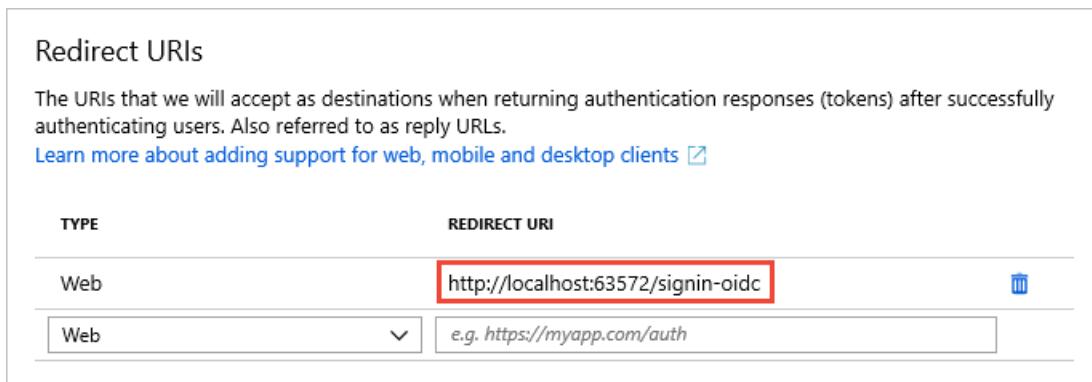
1. Navigate to your app registration in the Azure portal.
2. In the Manage section, select the **Authentication** setting.
3. Under **Advanced settings**, in the **Implicit grant** section, select the check boxes to enable access tokens and ID tokens, as shown in the following image:



## Update the port used by localhost

When you run the sample, you may find that you need to update the redirect URI specified in your app registration to use the */localhost* port assigned at runtime. To update the redirect URI to use the assigned port, follow these steps:

1. Navigate to your app registration in the Azure portal.
2. In the Manage section, select the **Authentication** setting.
3. Under **Redirect URIs**, edit the port to match that used by the sample application, as shown in the following image:



The screenshot shows the 'Redirect URIs' section of an Azure app registration. It includes a descriptive text about what URIs are for, a link to learn more, and a table with two entries. The first entry is highlighted with a red box around its URL column.

Type	Redirect URI	Action
Web	http://localhost:63572/signin-oidc	Delete
Web	e.g. https://myapp.com/auth	Add

## Next steps

- To learn more about the Microsoft identity platform, see [Microsoft identity platform](#).
- To learn more about RBAC roles for Azure storage, see [Manage access rights to storage data with RBAC](#).
- To learn about using managed identities for Azure resources with Azure Storage, see [Authenticate access to blobs and queues with Azure Active Directory and managed identities for Azure Resources](#).

# Manage storage account access keys

4/16/2020 • 4 minutes to read • [Edit Online](#)

When you create a storage account, Azure generates two 512-bit storage account access keys. 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

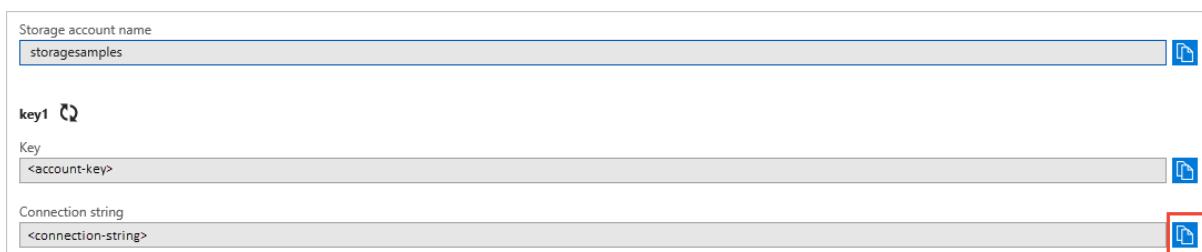
Your storage account access keys are similar to a root password for your storage account. Always be careful to protect your access keys. Use Azure Key Vault to manage and rotate your keys securely. Avoid distributing access keys to other users, hard-coding them, or saving them anywhere in plain text that is accessible to others. Rotate your keys if you believe they may have been compromised.

If possible, use Azure Active Directory (Azure AD) to authorize requests to Blob and Queue storage instead of Shared Key. Azure AD provides superior security and ease of use over Shared Key. For more information about authorizing access to data with Azure AD, see [Authorize access to Azure blobs and queues using Azure Active Directory](#).

## View access keys and connection string

To view and copy your storage account access keys or connection string from the Azure portal:

1. Navigate to the [Azure portal](#).
2. Locate your storage account.
3. Under **Settings**, select **Access keys**. Your account access keys appear, as well as the complete connection string for each key.
4. Find the **Key** value under **key1**, and click the **Copy** button to copy the account key.
5. Alternately, you can copy the entire connection string. Find the **Connection string** value under **key1**, and click the **Copy** button to copy the connection string.



You can use either key 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 RBAC role that includes the **Microsoft.Storage/storageAccounts/listkeys/action**. Some built-in RBAC 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 RBAC 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](#).

Follow this process to rotate your storage account keys:

1. Update the connection strings in your application code to use the secondary key.
2. Regenerate the primary access key for your storage account. On the **Access Keys** blade in the Azure portal, click **Regenerate Key1**, and then click **Yes** to confirm that you want to generate a new key.
3. Update the connection strings in your code to reference the new primary access key.
4. Regenerate the secondary access key in the same manner.

### NOTE

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 RBAC role that includes the **Microsoft.Storage/storageAccounts/regeneratekey/action**. Some built-in RBAC 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 RBAC roles, and Azure AD roles](#). For detailed information about built-in RBAC roles for Azure Storage, see the **Storage** section in [Azure built-in roles for Azure RBAC](#).

## Next steps

- [Azure storage account overview](#)
- [Create a storage account](#)

# Configure Azure Storage connection strings

12/23/2019 • 9 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 Azure storage emulator.
- Access a storage account in Azure.
- Access specified resources in Azure via a shared access signature (SAS).

## Protect your access keys

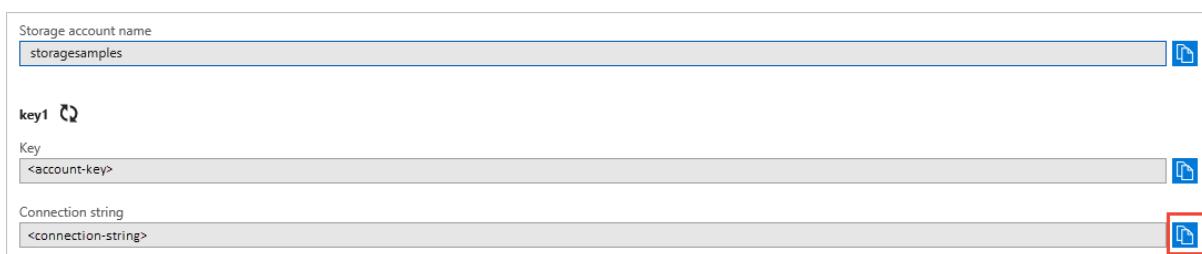
Your storage account access keys are similar to a root password for your storage account. Always be careful to protect your access keys. Use Azure Key Vault to manage and rotate your keys securely. Avoid distributing access keys to other users, hard-coding them, or saving them anywhere in plain text that is accessible to others. Rotate your keys if you believe they may have been compromised.

If possible, use Azure Active Directory (Azure AD) to authorize requests to Blob and Queue storage instead of Shared Key. Azure AD provides superior security and ease of use over Shared Key. For more information about authorizing access to data with Azure AD, see [Authorize access to Azure blobs and queues using Azure Active Directory](#).

## View and copy a connection string

To view and copy your storage account access keys or connection string from the Azure portal:

1. Navigate to the [Azure portal](#).
2. Locate your storage account.
3. Under **Settings**, select **Access keys**. Your account access keys appear, as well as the complete connection string for each key.
4. Find the **Key** value under **key1**, and click the **Copy** button to copy the account key.
5. Alternately, you can copy the entire connection string. Find the **Connection string** value under **key1**, and click the **Copy** button to copy the connection string.



You can use either key 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 RBAC role that includes the **Microsoft.Storage/storageAccounts/listkeys/action**. Some built-in RBAC 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](#)

RBAC 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](#).

## 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 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 the storage emulator

The storage 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 storage emulator. They are:

```
Account name: devstoreaccount1
Account key: Eby8vdM02xNOcqFlqUwJPLl1mEt1CDXJ10UzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==
```

### NOTE

The authentication key supported by the storage 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 storage emulator. You should not use the development account with production data.

The storage 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 a shortcut

The easiest way to connect to the storage emulator 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 to the storage emulator in an [app.config](#) file:

```
<appSettings>
 <add key="StorageConnectionString" value="UseDevelopmentStorage=true" />
</appSettings>
```

### Connect to the emulator account using the well-known account name and key

To create a connection string that references the emulator account name and key, you must specify the endpoints for each of the services you wish to use from the emulator in the connection string. This is necessary so that the connection string will reference the emulator endpoints, which are different than those for a production storage account. For example, the value of your connection string will look like this:

```
DefaultEndpointsProtocol=http;AccountName=devstoreaccount1;
AccountKey=Eby8vdM02xN0cqFlqUwJPLlmEtlCDXJ1OUzFT50uSRZ6IFsuFq2UVrCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==;
BlobEndpoint=http://127.0.0.1:10000/devstoreaccount1;
TableEndpoint=http://127.0.0.1:10002/devstoreaccount1;
QueueEndpoint=http://127.0.0.1:10001/devstoreaccount1;
```

This value is identical to the shortcut shown above, `UseDevelopmentStorage=true`.

#### Specify an HTTP proxy

You can also specify an HTTP proxy to use when you're testing your service against the storage emulator. This can be useful for observing HTTP requests and responses while you're debugging operations against the storage services. To specify a proxy, add the `DevelopmentStorageProxyUri` option to the connection string, and set its value to the proxy URI. For example, here is a connection string that points to the storage emulator and configures an HTTP proxy:

```
UseDevelopmentStorage=true;DevelopmentStorageProxyUri=http://myProxyUri
```

For more information about the storage emulator, see [Use the Azure storage emulator for development and testing](#).

## 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=9aCzs76n0E7y5BpEi2GvsSv433BZa221eD0ZXX%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=9aCzs76n0E7y5BpEi2GvsSv433BZa221eD0ZXX%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.

#### IMPORTANT

Service endpoint values in your connection strings must be well-formed URLs, including `https://` (recommended) or `http://`. Because Azure Storage does not yet support HTTPS for custom domains, you *must* specify `http://` for any endpoint URI that points to a custom domain.

#### 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 Azure storage emulator for development and testing](#)
- [Azure Storage explorers](#)
- [Using Shared Access Signatures \(SAS\)](#)

# Call REST API operations with Shared Key authorization

2/28/2020 • 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 `http://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:1dVlYJWWJAOSHTCPGiwdX1rOS8B4fenYP/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 for 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{n1}\n{n5}\n\n\n{n2}\n\n\n{n3}{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\ncontosorest\nncomp: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\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:uzvwZN1WUIv2LYC6e3En10/7EIQJ5X9KtFQqrZkxi6s=
```

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](#)

# Create a user delegation SAS for a container or blob with PowerShell

3/18/2020 • 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*.
- With the storage account key. Both a *service SAS* and an *account SAS* are signed with the storage account key.

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:

- 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`.
- 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 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 RBAC permissions for creating a user delegation SAS, see the **Assign permissions with RBAC** section in [Create a user delegation SAS](#).

If you do not have sufficient permissions to assign RBAC 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 [Built-in roles for Azure resources](#).

## 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%3A
00%3A00Z&skbs=b&skv=2018-11-09&se=2019-08-07T07%3A00%3A00Z&sp=rwdl
```

### 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 RBAC 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

12/18/2019 • 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*.
- With the storage account key. Both a *service SAS* and an *account SAS* are signed with the storage account key.

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 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 RBAC 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 [Built-in roles for Azure resources](#).

## 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&sks=b&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 URL 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&sks=b&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 RBAC 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 .NET

12/18/2019 • 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*.
- With the storage account key. Both a *service SAS* and an *account SAS* are signed with the storage account key.

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 Storage client library for .NET.

## 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 RBAC 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 RBAC role that grants access to blob data in Azure Storage. For information about assigning permissions via RBAC, see the section titled [Assign RBAC roles for access rights](#) in [Authorize access to Azure blobs and queues using Azure Active Directory](#).

## 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 from Azure Storage, see the section titled [Authenticate with the Azure Identity library](#) in [Authorize access to blobs and queues with Azure Active Directory and managed identities for Azure Resources](#).

## Add using directives

Add the following `using` directives to your code to use the Azure Identity and Azure Storage client libraries.

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

## 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.

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://{}.{}, 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);
```

## Create the SAS token

The following code snippet shows create a new [BlobSasBuilder](#) and provide the parameters for the user delegation SAS. The snippet then calls the [ToSasQueryParameters](#) to get the SAS token string. Finally, the code builds the complete URI, including the resource address and SAS token.

```
// Create a SAS token that's valid for one hour.
BlobSasBuilder sasBuilder = new BlobSasBuilder()
{
 BlobContainerName = containerName,
 BlobName = blobName,
 Resource = "b",
 StartsOn = DateTimeOffset.UtcNow,
 ExpiresOn = DateTimeOffset.UtcNow.AddHours(1)
};

// Specify read permissions for the SAS.
sasBuilder.SetPermissions(BlobSasPermissions.Read);

// Use the key to get the SAS token.
string sasToken = sasBuilder.ToSasQueryParameters(key, accountName).ToString();

// Construct the full URI, including the SAS token.
UriBuilder fullUri = new UriBuilder()
{
 Scheme = "https",
 Host = string.Format("{0}.blob.core.windows.net", accountName),
 Path = string.Format("{0}/{1}", containerName, blobName),
 Query = sasToken
};
```

## Example: Get a user delegation SAS

The following example method shows the complete code for authenticating the security principal and creating the

user delegation SAS:

```
async static Task<Uri> GetUserDelegationSasBlob(string accountName, string containerName, string blobName)
{
 // 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 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);
 Console.WriteLine();

 // Create a SAS token that's valid for one hour.
 BlobSasBuilder sasBuilder = new BlobSasBuilder()
 {
 BlobContainerName = containerName,
 BlobName = blobName,
 Resource = "b",
 StartsOn = DateTimeOffset.UtcNow,
 ExpiresOn = DateTimeOffset.UtcNow.AddHours(1)
 };

 // Specify read permissions for the SAS.
 sasBuilder.SetPermissions(BlobSasPermissions.Read);

 // Use the key to get the SAS token.
 string sasToken = sasBuilder.ToSasQueryParameters(key, accountName).ToString();

 // Construct the full URI, including the SAS token.
 UriBuilder fullUri = new UriBuilder()
 {
 Scheme = "https",
 Host = string.Format("{0}.blob.core.windows.net", accountName),
 Path = string.Format("{0}/{1}", containerName, blobName),
 Query = sasToken
 };

 Console.WriteLine("User delegation SAS URI: {0}", fullUri);
 Console.WriteLine();
 return fullUri.Uri;
}
```

## Example: Read a blob with a user delegation SAS

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 ReadBlobWithSasAsync(Uri sasUri)
{
 // Try performing blob operations using the 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
 {
 // 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;
 }
 }
}
}

```

## 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 with .NET

3/12/2020 • 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*.
- With the storage account key. Both a *service SAS* and an *account SAS* are signed with the storage account key.

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 .NET](#).

## Create a service SAS for a blob container

To create a service SAS for a container, call the [CloudBlobContainer.GetSharedAccessSignature](#) method.

The following code example creates a SAS on 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.

```

private static string GetContainerSasUri(CloudBlobContainer container, string storedPolicyName = null)
{
 string sasContainerToken;

 // If no stored policy is specified, create a new access policy and define its constraints.
 if (storedPolicyName == null)
 {
 // Note that the SharedAccessBlobPolicy class is used both to define the parameters of an ad hoc SAS,
 and
 // to construct a shared access policy that is saved to the container's shared access policies.
 SharedAccessBlobPolicy adHocPolicy = new SharedAccessBlobPolicy()
 {
 // When the start time for the SAS is omitted, the start time is assumed to be the time when the
 storage service receives the request.
 // Omitting the start time for a SAS that is effective immediately helps to avoid clock skew.
 SharedAccessExpiryTime = DateTime.UtcNow.AddHours(24),
 Permissions = SharedAccessBlobPermissions.Write | SharedAccessBlobPermissions.List
 };
 }

 // Generate the shared access signature on the container, setting the constraints directly on the
 signature.
 sasContainerToken = container.GetSharedAccessSignature(adHocPolicy, null);

 Console.WriteLine("SAS for blob container (ad hoc): {0}", sasContainerToken);
 Console.WriteLine();
}

else
{
 // Generate the shared access signature on the container. In this case, all of the constraints for the
 // shared access signature are specified on the stored access policy, which is provided by name.
 // It is also possible to specify some constraints on an ad hoc SAS and others on the stored access
 policy.
 sasContainerToken = container.GetSharedAccessSignature(null, storedPolicyName);

 Console.WriteLine("SAS for blob container (stored access policy): {0}", sasContainerToken);
 Console.WriteLine();
}

// Return the URI string for the container, including the SAS token.
return container.Uri + sasContainerToken;
}

```

## Create a service SAS for a blob

To create a service SAS for a blob, call the [CloudBlob.GetSharedAccessSignature](#) method.

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.

```

private static string GetBlobSasUri(CloudBlobContainer container, string blobName, string policyName = null)
{
 string sasBlobToken;

 // Get a reference to a blob within the container.
 // Note that the blob may not exist yet, but a SAS can still be created for it.
 CloudBlockBlob blob = container.GetBlockBlobReference(blobName);

 if (policyName == null)
 {
 // Create a new access policy and define its constraints.
 // Note that the SharedAccessBlobPolicy class is used both to define the parameters of an ad hoc SAS,
 and
 // to construct a shared access policy that is saved to the container's shared access policies.
 SharedAccessBlobPolicy adHocSAS = new SharedAccessBlobPolicy()
 {
 // When the start time for the SAS is omitted, the start time is assumed to be the time when the
 storage service receives the request.
 // Omitting the start time for a SAS that is effective immediately helps to avoid clock skew.
 SharedAccessExpiryTime = DateTime.UtcNow.AddHours(24),
 Permissions = SharedAccessBlobPermissions.Read | SharedAccessBlobPermissions.Write |
 SharedAccessBlobPermissions.Create
 };
 }

 // Generate the shared access signature on the blob, setting the constraints directly on the
 signature.
 sasBlobToken = blob.GetSharedAccessSignature(adHocSAS);

 Console.WriteLine("SAS for blob (ad hoc): {0}", sasBlobToken);
 Console.WriteLine();
}

else
{
 // Generate the shared access signature on the blob. In this case, all of the constraints for the
 // shared access signature are specified on the container's stored access policy.
 sasBlobToken = blob.GetSharedAccessSignature(null, policyName);

 Console.WriteLine("SAS for blob (stored access policy): {0}", sasBlobToken);
 Console.WriteLine();
}

// Return the URI string for the container, including the SAS token.
return blob.Uri + sasBlobToken;
}

```

## 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](#)

## Next steps

- [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#)
- [Create a service SAS](#)

# Create an account SAS with .NET

3/12/2020 • 2 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*.
- With the storage account key. Both a *service SAS* and an *account SAS* are signed with the storage account key.

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

To create an account SAS for a container, call the [CloudStorageAccount.GetSharedAccessSignature](#) method.

The following code example creates an account SAS that is valid for the Blob and File services, and gives the client permissions read, write, and list permissions to access service-level APIs. The account SAS restricts the protocol to HTTPS, so the request must be made with HTTPS. Remember to replace placeholder values in angle brackets with your own values:

```
static string GetAccountSASToken()
{
 // To create the account SAS, you need to use Shared Key credentials. Modify for your account.
 const string ConnectionString = "DefaultEndpointsProtocol=https;AccountName=<storage-account>;AccountKey=<account-key>";
 CloudStorageAccount storageAccount = CloudStorageAccount.Parse(ConnectionString);

 // Create a new access policy for the account.
 SharedAccessAccountPolicy policy = new SharedAccessAccountPolicy()
 {
 Permissions = SharedAccessAccountPermissions.Read | SharedAccessAccountPermissions.Write |
 SharedAccessAccountPermissions.List,
 Services = SharedAccessAccountServices.Blob | SharedAccessAccountServices.File,
 ResourceTypes = SharedAccessAccountResourceTypes.Service,
 SharedAccessExpiryTime = DateTime.UtcNow.AddHours(24),
 Protocols = SharedAccessProtocol.HttpsOnly
 };

 // Return the SAS token.
 return storageAccount.GetSharedAccessSignature(policy);
}
```

## 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. Remember to replace placeholder values in angle

brackets with your own values:

```
static void UseAccountSAS(string sasToken)
{
 // Create new storage credentials using the SAS token.
 StorageCredentials accountSAS = new StorageCredentials(sasToken);
 // Use these credentials and the account name to create a Blob service client.
 CloudStorageAccount accountWithSAS = new CloudStorageAccount(accountSAS, "<storage-account>",
 endpointSuffix: null, useHttps: true);
 CloudBlobClient blobClientWithSAS = accountWithSAS.CreateCloudBlobClient();

 // Now set the service properties for the Blob client created with the SAS.
 blobClientWithSAS.SetServiceProperties(new ServiceProperties()
 {
 HourMetrics = new MetricsProperties()
 {
 MetricsLevel = MetricsLevel.ServiceAndApi,
 RetentionDays = 7,
 Version = "1.0"
 },
 MinuteMetrics = new MetricsProperties()
 {
 MetricsLevel = MetricsLevel.ServiceAndApi,
 RetentionDays = 7,
 Version = "1.0"
 },
 Logging = new LoggingProperties()
 {
 LoggingOperations = LoggingOperations.All,
 RetentionDays = 14,
 Version = "1.0"
 }
 });
}

// The permissions granted by the account SAS also permit you to retrieve service properties.
ServiceProperties serviceProperties = blobClientWithSAS.GetServiceProperties();
Console.WriteLine(serviceProperties.HourMetrics.MetricsLevel);
Console.WriteLine(serviceProperties.HourMetrics.RetentionDays);
Console.WriteLine(serviceProperties.HourMetrics.Version);
}
```

## Next steps

- [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#)
- [Create an account SAS](#)

# Define a stored access policy with .NET

8/13/2019 • 2 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 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.

## Create a stored access policy

The following code creates 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.

```
private static async Task CreateStoredAccessPolicyAsync(CloudBlobContainer container, string policyName)
{
 // Create a new stored access policy and define its constraints.
 // The access policy provides create, write, read, list, and delete permissions.
 SharedAccessBlobPolicy sharedPolicy = new SharedAccessBlobPolicy()
 {
 // When the start time for the SAS is omitted, the start time is assumed to be the time when Azure Storage receives the request.
 SharedAccessExpiryTime = DateTime.UtcNow.AddHours(24),
 Permissions = SharedAccessBlobPermissions.Read | SharedAccessBlobPermissions.List |
 SharedAccessBlobPermissions.Write | SharedAccessBlobPermissions.Create |
 SharedAccessBlobPermissions.Delete
 };

 // Get the container's existing permissions.
 BlobContainerPermissions permissions = await container.GetPermissionsAsync();

 // Add the new policy to the container's permissions, and set the container's permissions.
 permissions.SharedAccessPolicies.Add(policyName, sharedPolicy);
 await container.SetPermissionsAsync(permissions);
}
```

## See also

- Grant limited access to Azure Storage resources using shared access signatures (SAS)
- Define a stored access policy

# Check the encryption status of a blob

12/2/2019 • 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 Storage Blob Overview page for a blob named 'blob1.txt'. The top navigation bar includes Save, Discard, Download, Refresh, Delete, Change tier, Acquire lease, and Break lease buttons. Below the navigation bar, there are tabs for Overview, Snapshots, Edit, and Generate SAS, with 'Overview' being the active tab. The main content area displays blob properties in a table format. The 'Server Encrypted' property is highlighted with a red box. Other visible properties include URL (https://storagesamples.blob.core.windows.net/sampl...), 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 (-), and COPY COMPLETION TIME (-).

Properties	Values
URL	https://storagesamples.blob.core.windows.net/sampl...
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

3/15/2020 • 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

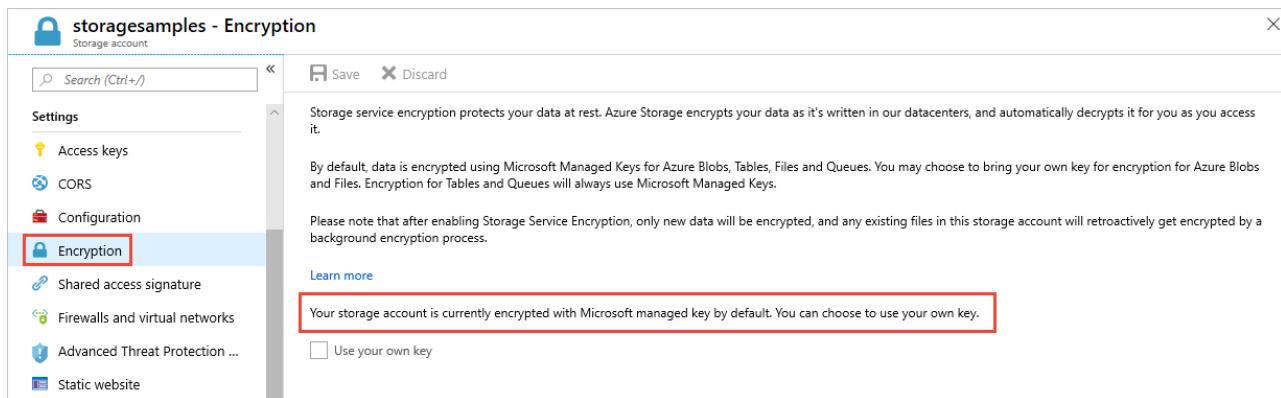
To determine whether a storage account is using Microsoft-managed keys or customer-managed keys for encryption, use one of the following approaches.

- [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:

The screenshot shows the 'Encryption' blade for the 'storagesamples' storage account. On the left, there's a navigation menu with options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Data transfer, Events, and Storage Explorer (preview). The main area has a search bar and Save/Discard buttons. A checkbox labeled 'Use your own key' is checked and highlighted with a red box. Below it, there's a section for 'Encryption key' with two options: 'Enter key URI' (selected) and 'Select from Key Vault'. A 'Key URI \*' input field contains the value 'https://storage-key-vault-sample.vault.azure.net/keys/storage-sample-key/7e44c8e00fc42cdb1a84c63e4bd9ff6'. A note below the input field states: '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.' with a 'Learn more' link.

## Next steps

- [Azure Storage encryption for data at rest](#)
- [Use customer-managed keys with Azure Key Vault to manage Azure Storage encryption](#)

# Configure customer-managed keys with Azure Key Vault by using the Azure portal

4/16/2020 • 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 supply customer-managed keys to use for encryption of blob and file data.

Customer-managed keys must be stored in an Azure Key Vault. You can either create your own keys and store them in a key vault, or you can use the Azure Key Vault APIs to generate keys. The storage account and the key vault must be in the same region, but they can be in different subscriptions. For more information about Azure Storage encryption and key management, see [Azure Storage encryption for data at rest](#). For more information about Azure Key Vault, see [What is Azure Key Vault?](#)

This article shows how to configure an Azure Key Vault with customer-managed keys using the [Azure portal](#). To learn how to create a key vault using the Azure portal, see [Quickstart: Set and retrieve a secret from Azure Key Vault using the Azure portal](#).

## Configure Azure Key Vault

Using customer-managed keys with Azure Storage encryption requires that two properties be set on the key vault, **Soft Delete** and **Do Not Purge**. These properties are not enabled by default, but can be enabled using either PowerShell or Azure CLI on a new or existing key vault.

To learn how to enable these properties on an existing key vault, see the sections titled **Enabling soft-delete** and **Enabling Purge Protection** in one of the following articles:

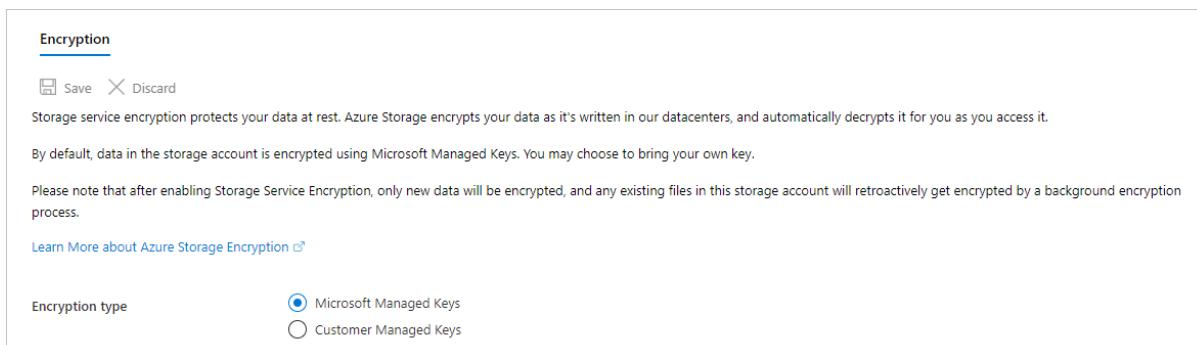
- [How to use soft-delete with PowerShell](#).
- [How to use soft-delete with CLI](#).

Only 2048-bit RSA and RSA-HSM keys are supported with Azure Storage encryption. For more information about keys, see [Key Vault keys](#) in [About Azure Key Vault keys, secrets and certificates](#).

## Enable customer-managed keys

To enable customer-managed keys in the Azure portal, follow these steps:

1. Navigate to your storage account.
2. On the **Settings** blade for the storage account, click **Encryption**. Select the **Customer Managed Keys** option, as shown in the following image.



# Specify a key

After you enable customer-managed keys, you'll have the opportunity to specify a key to associate with the storage account.

## Specify a key as a URI

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.

The screenshot shows the 'Keys' blade in the Azure portal. A key named '17bf9182bb694f109b8dc6d1e9b69f29' is selected. The 'Key Identifier' field contains the value '<key-uri>'. Other properties shown include Key Type (RSA), RSA Key Size (2048), and creation and update dates.

3. In the **Encryption** settings for your storage account, choose the **Enter key URI** option.
4. Paste the URI that you copied into the **Key URI** field.

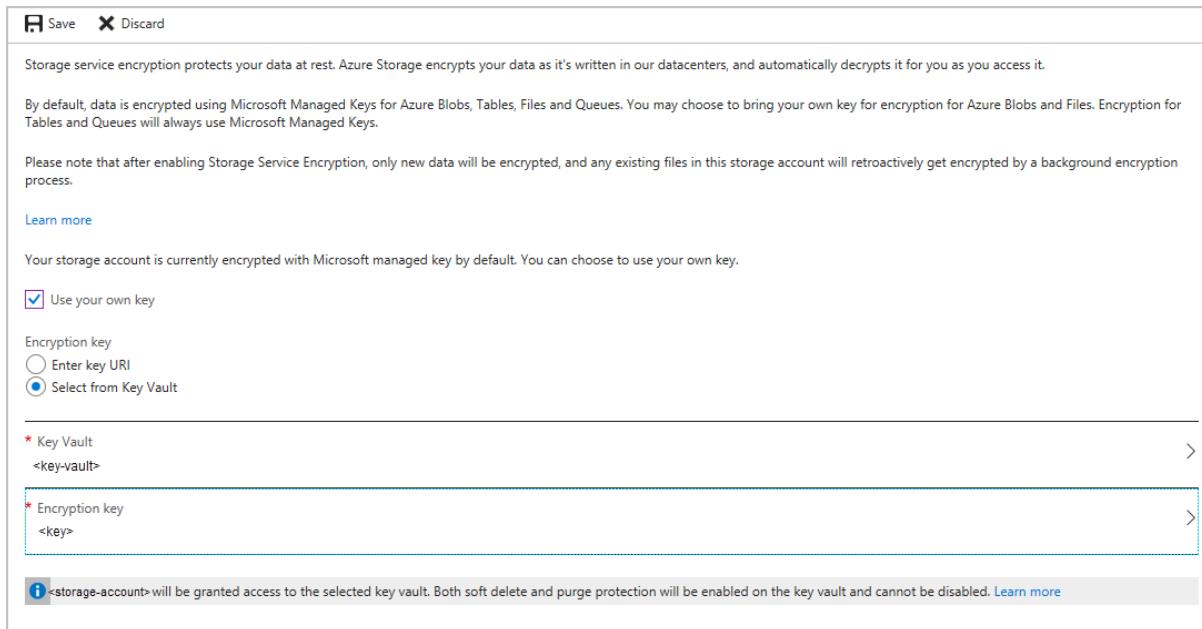
The screenshot shows the 'Encryption' blade. The 'Use your own key' checkbox is checked. Under 'Encryption key', the 'Enter key URI' radio button is selected. The 'Key URI' field contains the value '<key-uri>'. A note at the bottom states: '<storage-account> will be granted access to the selected key vault. Both soft delete and purge protection will be enabled on the key vault and cannot be disabled.' with a 'Learn more' link.

5. Specify the subscription that contains the key vault.
6. Save your changes.

## Specify a key from a key vault

To specify a key from a key vault, first make sure that you have a key vault that contains a key. To specify a key from a key vault, follow these steps:

1. Choose the **Select from Key Vault** option.
2. Select the key vault containing the key you want to use.
3. Select the key from the key vault.



4. Save your changes.

## Update the key version

When you create a new version of a key, update the storage account to use the new version. Follow these steps:

1. Navigate to your storage account and display the **Encryption** settings.
2. Enter the URI for the new key version. Alternately, you can select the key vault and the key again to update the version.
3. Save your changes.

## Use a different key

To change the key used for Azure Storage encryption, follow these steps:

1. Navigate to your storage account and display the **Encryption** settings.
2. Enter the URI for the new key. Alternately, you can select the key vault and choose a new key.
3. Save your changes.

## Disable customer-managed keys

When you disable customer-managed keys, your storage account is once again encrypted with Microsoft-managed keys. To disable customer-managed keys, 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](#)

- [What is Azure Key Vault?](#)

# Configure customer-managed keys with Azure Key Vault by using PowerShell

4/16/2020 • 4 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 supply customer-managed keys to use for encryption of blob and file data.

Customer-managed keys must be stored in an Azure Key Vault. You can either create your own keys and store them in a key vault, or you can use the Azure Key Vault APIs to generate keys. The storage account and the key vault must be in the same region, but they can be in different subscriptions. For more information about Azure Storage encryption and key management, see [Azure Storage encryption for data at rest](#). For more information about Azure Key Vault, see [What is Azure Key Vault?](#)

This article shows how to configure an Azure Key Vault with customer-managed keys using PowerShell. To learn how to create a key vault using Azure CLI, see [Quickstart: Set and retrieve a secret from Azure Key Vault using PowerShell](#).

## Assign an identity to the storage account

To enable customer-managed keys for your 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 key vault.

To assign a managed identity using PowerShell, call [Set-AzStorageAccount](#). Remember to replace the placeholder values in brackets with your own values.

```
$storageAccount = Set-AzStorageAccount -ResourceGroupName <resource_group> `
 -Name <storage-account> `
 -AssignIdentity
```

For more information about configuring system-assigned managed identities with PowerShell, see [Configure managed identities for Azure resources on an Azure VM using PowerShell](#).

## Create a new key vault

To create a new key vault using PowerShell, call [New-AzKeyVault](#). The key vault that you use to store customer-managed keys for Azure Storage encryption must have two key protection settings enabled, **Soft Delete** and **Do Not Purge**.

Remember to replace the placeholder values in brackets with your own values.

```
$keyVault = New-AzKeyVault -Name <key-vault> `
 -ResourceGroupName <resource_group> `
 -Location <location> `
 -EnableSoftDelete `
 -EnablePurgeProtection
```

To learn how to enable **Soft Delete** and **Do Not Purge** on an existing key vault with PowerShell, see the sections titled **Enabling soft-delete** and **Enabling Purge Protection** in [How to use soft-delete with PowerShell](#).

## Configure the key vault access policy

Next, configure the access policy for the key vault so that the storage account has permissions to access it. In this step, you'll use the managed identity that you previously assigned to the storage account.

To set the access policy for the key vault, call [Set-AzKeyVaultAccessPolicy](#). Remember to replace the placeholder values in brackets with your own values and to use the variables defined in the previous examples.

```
Set-AzKeyVaultAccessPolicy `
 -VaultName $keyVault.VaultName `
 -ObjectId $storageAccount.Identity.PrincipalId `
 -PermissionsToKeys wrapkey,unwrapkey,get
```

## Create a new key

Next, create a new key in the key vault. To create a new key, call [Add-AzKeyVaultKey](#). Remember to replace the placeholder values in brackets with your own values and to use the variables defined in the previous examples.

```
$key = Add-AzKeyVaultKey -VaultName $keyVault.VaultName -Name <key> -Destination 'Software'
```

Only 2048-bit RSA and RSA-HSM keys are supported with Azure Storage encryption. For more information about keys, see [Key Vault keys](#) in [About Azure Key Vault keys, secrets and certificates](#).

## Configure encryption with customer-managed keys

By default, Azure Storage encryption uses Microsoft-managed keys. In this step, configure your Azure Storage account to use customer-managed keys and specify the key to associate with the storage account.

Call [Set-AzStorageAccount](#) to update the storage account's encryption settings, as shown in the following example. Include the **-KeyvaultEncryption** option to enable customer-managed keys for the storage account. Remember to replace the placeholder values in brackets with your own values and to use the variables defined in the previous examples.

```
Set-AzStorageAccount -ResourceGroupName $storageAccount.ResourceGroupName `
 -AccountName $storageAccount.StorageAccountName `
 -KeyvaultEncryption `
 -KeyName $key.Name `
 -KeyVersion $key.Version `
 -KeyVaultUri $keyVault.VaultUri
```

## Update the key version

When you create a new version of a key, you'll need to update the storage account to use the new version. First, call [Get-AzKeyVaultKey](#) to get the latest version of the key. Then call [Set-AzStorageAccount](#) to update the storage account's encryption settings to use the new version of the key, as shown in the previous section.

## Use a different key

To change the key used for Azure Storage encryption, call [Set-AzStorageAccount](#) as shown in [Configure encryption with customer-managed keys](#) and provide the new key name and version. If the new key is in a different key vault, also update the key vault URL.

## Revoke customer-managed keys

If you believe that a key may have been compromised, you can revoke customer-managed keys by removing the key vault access policy. To revoke a customer-managed key, call the [Remove-AzKeyVaultAccessPolicy](#) command, as shown in the following example. Remember to replace the placeholder values in brackets with your own values and to use the variables defined in the previous examples.

```
Remove-AzKeyVaultAccessPolicy -VaultName $keyVault.VaultName `
-ObjectId $storageAccount.Identity.PrincipalId `
```

## Disable customer-managed keys

When you disable customer-managed keys, your storage account is once again encrypted with Microsoft-managed keys. To disable customer-managed keys, call [Set-AzStorageAccount](#) with the `-StorageEncryption` option, as shown in the following example. Remember to replace the placeholder values in brackets with your own values and to use the variables defined in the previous examples.

```
Set-AzStorageAccount -ResourceGroupName $storageAccount.ResourceGroupName `
-AccountName $storageAccount.StorageAccountName `
-StorageEncryption
```

## Next steps

- [Azure Storage encryption for data at rest](#)
- [What is Azure Key Vault?](#)

# Configure customer-managed keys with Azure Key Vault by using Azure CLI

4/16/2020 • 5 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 supply customer-managed keys to use for encryption of blob and file data.

Customer-managed keys must be stored in an Azure Key Vault. You can either create your own keys and store them in a key vault, or you can use the Azure Key Vault APIs to generate keys. The storage account and the key vault must be in the same region, but they can be in different subscriptions. For more information about Azure Storage encryption and key management, see [Azure Storage encryption for data at rest](#). For more information about Azure Key Vault, see [What is Azure Key Vault?](#)

This article shows how to configure an Azure Key Vault with customer-managed keys using Azure CLI. To learn how to create a key vault using Azure CLI, see [Quickstart: Set and retrieve a secret from Azure Key Vault using Azure CLI](#).

## Assign an identity to the storage account

To enable customer-managed keys for your 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 key vault.

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 account set --subscription <subscription-id>

az storage account update \
 --name <storage-account> \
 --resource-group <resource_group> \
 --assign-identity
```

For more information about configuring system-assigned managed identities with Azure CLI, see [Configure managed identities for Azure resources on an Azure VM using Azure CLI](#).

## Create a new key vault

The key vault that you use to store customer-managed keys for Azure Storage encryption must have two key protection settings enabled, **Soft Delete** and **Do Not Purge**. To create a new key vault using PowerShell or Azure CLI with these settings enabled, execute the following commands. Remember to replace the placeholder values in brackets with your own values.

To create a new key vault using Azure CLI, call [az keyvault create](#). Remember to replace the placeholder values in brackets with your own values.

```
az keyvault create \
 --name <key-vault> \
 --resource-group <resource_group> \
 --location <region> \
 --enable-soft-delete \
 --enable-purge-protection
```

To learn how to enable **Soft Delete** and **Do Not Purge** on an existing key vault with Azure CLI, see the sections titled **Enabling soft-delete** and **Enabling Purge Protection** in [How to use soft-delete with CLI](#).

## Configure the key vault access policy

Next, configure the access policy for the key vault so that the storage account has permissions to access it. In this step, you'll use the managed identity that you previously assigned to the storage account.

To set the access policy for the key vault, call [az keyvault set-policy](#). 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 set-policy \
 --name <key-vault> \
 --resource-group <resource_group>
 --object-id $storage_account_principal \
 --key-permissions get unwrapKey wrapKey
```

## Create a new key

Next, create a key in the key vault. To create a key, call [az keyvault key create](#). Remember to replace the placeholder values in brackets with your own values.

```
az keyvault key create
 --name <key> \
 --vault-name <key-vault>
```

Only 2048-bit RSA and RSA-HSM keys are supported with Azure Storage encryption. For more information about keys, see **Key Vault keys** in [About Azure Key Vault keys, secrets and certificates](#).

## Configure encryption with customer-managed keys

By default, Azure Storage encryption uses Microsoft-managed keys. Configure your Azure Storage account for customer-managed keys and specify the key to associate with the storage account.

To update the storage account's encryption settings, call [az storage account update](#), 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 storage account. The example also queries for the key vault URI and the latest key version, both of which values are needed to associate the key with the storage account. Remember to replace the placeholder values in brackets with your own values.

```
key_vault_uri=$(az keyvault show \
--name <key-vault> \
--resource-group <resource_group> \
--query properties.vaultUri \
--output tsv)
key_version=$(az keyvault key list-versions \
--name <key> \
--vault-name <key-vault> \
--query [-1].kid \
--output tsv | cut -d '/' -f 6)
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 $key_vault_uri
```

## Update the key version

When you create a new version of a key, you'll need to update the storage account 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 section.

## Use a different key

To change the key used for Azure Storage encryption, call [az storage account update](#) as shown in [Configure encryption with customer-managed keys](#) and provide the new key name and version. If the new key is in a different key vault, also update the key vault URI.

## Revoke customer-managed keys

If you believe that a key may have been compromised, you can revoke customer-managed keys by removing the key vault access policy. To revoke a customer-managed key, call the [az keyvault delete-policy](#) command, as shown in the following example. Remember to replace the placeholder values in brackets with your own values and to use the variables defined in the previous examples.

```
az keyvault delete-policy \
--name <key-vault> \
--object-id $storage_account_principal
```

## Disable customer-managed keys

When you disable customer-managed keys, your storage account is once again encrypted with Microsoft-managed keys. To disable customer-managed keys, call [az storage account update](#) and set the `--encryption-key-source` parameter to `Microsoft.Storage`, as shown in the following example. Remember to replace the placeholder values in brackets with your own values and to use the variables defined in the previous examples.

```
az storage account update
--name <storage-account> \
--resource-group <resource_group> \
--encryption-key-source Microsoft.Storage
```

## Next steps

- [Azure Storage encryption for data at rest](#)
- [What is Azure Key Vault?](#)

# Specify a customer-provided key on a request to Blob storage with .NET

12/4/2019 • 2 minutes to read • [Edit Online](#)

Clients making requests against Azure Blob storage have the option to provide an 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 (preview) 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 from Azure Storage, see the section titled [Authenticate with the Azure Identity library](#) in [Authorize access to blobs and queues with Azure Active Directory and managed identities for Azure Resources](#).

## Example: Use a customer-provided key to upload a blob

The following example creates a customer-provided key and uses that key to upload a blob. The code uploads a block, then commits the block list to write the blob to Azure Storage.

```

async static Task UploadBlobWithClientKey(string accountName, string containerName,
 string blobName, Stream data, byte[] key)
{
 const string blobServiceEndpointSuffix = ".blob.core.windows.net";
 Uri accountUri = new Uri("https://" + accountName + blobServiceEndpointSuffix);

 // Specify the customer-provided key on the options for the client.
 BlobClientOptions options = new BlobClientOptions()
 {
 CustomerProvidedKey = new CustomerProvidedKey(key)
 };

 // Create a client object for the Blob service, including options.
 BlobServiceClient serviceClient = new BlobServiceClient(accountUri,
 new DefaultAzureCredential(), options);

 // Create a client object for the container.
 // The container client retains the credential and client options.
 BlobContainerClient containerClient = serviceClient.GetBlobContainerClient(containerName);

 // Create a new block blob client object.
 // The blob client retains the credential and client options.
 BlobClient blobClient = containerClient.GetBlobClient(blobName);

 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

- [Azure Storage encryption for data at rest](#)
- [Authorize access to blobs and queues with Azure Active Directory and managed identities for Azure Resources](#)

# Client-Side Encryption and Azure Key Vault for Microsoft Azure Storage

4/16/2020 • 13 minutes to read • [Edit Online](#)

## Overview

The [Azure 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.

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](#).

For client-side encryption with Java, see [Client-Side Encryption with Java for Microsoft Azure Storage](#).

## Encryption and decryption via the envelope technique

The processes of encryption and decryption follow the envelope technique.

### Encryption via the envelope technique

Encryption via the envelope technique works in the following way:

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 this 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 an asymmetric key pair or a symmetric key and can be managed locally or stored in Azure Key Vaults.

The 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 the Azure Storage service. The wrapped key along with some additional encryption metadata is either stored as metadata (on a blob) or interpolated with the encrypted data (queue messages and table entities).

### Decryption via the envelope technique

Decryption via the envelope technique works in the following way:

1. The client library assumes that the user is managing the key encryption key (KEK) either locally or in Azure Key Vaults. The user does not need to know the specific key that was used for encryption. Instead, a key resolver which 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 on the service.
3. The wrapped content encryption key (CEK) is then unwrapped (decrypted) using the key encryption key (KEK). Here again, the client library does not have access to KEK. It simply invokes the custom or Key Vault provider's unwrapping algorithm.
4. The content encryption key (CEK) is then used to decrypt the encrypted user data.

## Encryption Mechanism

The storage client library uses [AES](#) in order to encrypt user data. Specifically, [Cipher Block Chaining \(CBC\)](#) mode with AES. Each service works somewhat differently, so we will discuss each of them here.

## Blobs

The client library currently supports encryption of whole blobs only. Specifically, encryption is supported when users use the **UploadFrom** methods or the **OpenWrite** method. For downloads, both complete and range downloads are supported.

During encryption, the client library will generate a random Initialization Vector (IV) of 16 bytes, together with a random content encryption key (CEK) of 32 bytes, and perform 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 on the service.

### WARNING

If you are editing or uploading your own metadata for the blob, you need to ensure that this metadata is preserved. If you upload new metadata without this metadata, the wrapped CEK, IV, and other metadata will be lost and the blob content will never be retrievable again.

Downloading an encrypted blob involves retrieving the content of the entire blob using the **DownloadTo/BlobReadStream** convenience methods. The wrapped CEK is unwrapped and used together with the IV (stored as blob metadata in this case) to return the decrypted data to the users.

Downloading an arbitrary range (**DownloadRange** methods) 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.

## Queues

Since queue messages can be of any format, the client library defines a custom format that includes the Initialization Vector (IV) and the encrypted content encryption key (CEK) in the message text.

During encryption, the client library generates a random IV of 16 bytes along with a random CEK of 32 bytes and performs envelope encryption of the queue message text using this information. The wrapped CEK and some additional encryption metadata are then added to the encrypted queue message. This modified message (shown below) is stored on the service.

```
<MessageText>
{"EncryptedMessageContents": "6k0u8Rq1C3+M1Q04a1KLmWthWXSmHV3mEfxbAgP9QGTU++MKn2uPq3t2UjF1D06w", "EncryptionData"
 : {...}}</MessageText>
```

During decryption, the wrapped key is extracted from the queue message and unwrapped. The IV is also extracted from the queue message and used along with the unwrapped key to decrypt the queue message data. Note that the encryption metadata is small (under 500 bytes), so while it does count toward the 64KB limit for a queue message, the impact should be manageable.

## Tables

The client library supports encryption of entity properties for insert and replace operations.

#### **NOTE**

Merge is not currently supported. Since a subset of properties may have been encrypted previously using a different key, simply merging the new properties and updating the metadata will result in data loss. Merging either requires making extra service calls to read the pre-existing entity from the service, or using a new key per property, both of which are not suitable for performance reasons.

Table data encryption works as follows:

1. Users specify the properties to be encrypted.
2. The client library generates a random Initialization Vector (IV) of 16 bytes along with a random content encryption key (CEK) of 32 bytes for every entity, and performs envelope encryption on the individual properties to be encrypted by deriving a new IV per property. The encrypted property is stored as binary data.
3. The wrapped CEK and some additional encryption metadata are then stored as two additional reserved properties. The first reserved property (`_ClientEncryptionMetadata1`) is a string property that holds the information about IV, version, and wrapped key. The second reserved property (`_ClientEncryptionMetadata2`) is a binary property that holds the information about the properties that are encrypted. The information in this second property (`_ClientEncryptionMetadata2`) is itself encrypted.
4. Due to these additional reserved properties required for encryption, users may now have only 250 custom properties instead of 252. The total size of the entity must be less than 1 MB.

Note that only string properties can be encrypted. If other types of properties are to be encrypted, they must be converted to strings. The encrypted strings are stored on the service as binary properties, and they are converted back to strings after decryption.

For tables, in addition to the encryption policy, users must specify the properties to be encrypted. This can be done by either specifying an `[EncryptProperty]` attribute (for POCO entities that derive from `TableEntity`) or an encryption resolver in request options. An encryption resolver is a delegate that takes a partition key, row key, and property name and returns a Boolean that indicates whether that property should be encrypted. During encryption, the client library will use this information to decide whether a property should be encrypted while writing to the wire. The delegate also provides for the possibility of logic around how properties are encrypted. (For example, if X, then encrypt property A; otherwise encrypt properties A and B.) Note that it is not necessary to provide this information while reading or querying entities.

## **Batch Operations**

In batch operations, the same KEK will be used across all the rows in that batch operation because the client library only allows one options object (and hence one policy/KEK) per batch operation. However, the client library will internally generate a new random IV and random CEK per row in the batch. Users can also choose to encrypt different properties for every operation in the batch by defining this behavior in the encryption resolver.

## **Queries**

#### **NOTE**

Because the entities are encrypted, you cannot run queries that filter on an encrypted property. If you try, results will be incorrect, because the service would be trying to compare encrypted data with unencrypted data.

To perform query operations, you must specify a key resolver that is able to resolve all the keys in the result set. If an entity contained in the query result cannot be resolved to a provider, the client library will throw an error. For any query that performs server-side projections, the client library will add the special encryption metadata properties (`_ClientEncryptionMetadata1` and `_ClientEncryptionMetadata2`) by default to the selected columns.

# Azure Key Vault

Azure Key Vault helps safeguard cryptographic keys and secrets used by cloud applications and services. By using Azure Key Vault, users can encrypt keys and secrets (such as authentication keys, storage account keys, data encryption keys, .PFX files, and passwords) by using keys that are protected by hardware security modules (HSMs). For more information, see [What is Azure Key Vault?](#).

The storage client library uses the Key Vault core library in order to provide a common framework across Azure for managing keys. Users also get the additional benefit of using the Key Vault extensions library. The extensions library provides useful functionality around simple and seamless Symmetric/RSA local and cloud key providers as well as with aggregation and caching.

## Interface and dependencies

There are three Key Vault packages:

- Microsoft.Azure.KeyVault.Core contains the IKey and IKeyResolver. It is a small package with no dependencies. The storage client library for .NET defines it as a dependency.
- Microsoft.Azure.KeyVault contains the Key Vault REST client.
- Microsoft.Azure.KeyVault.Extensions contains extension code that includes implementations of cryptographic algorithms and an RSAKey and a SymmetricKey. It depends on the Core and KeyVault namespaces and provides functionality to define an aggregate resolver (when users want to use multiple key providers) and a caching key resolver. Although the storage client library does not directly depend on this package, if users wish to use Azure Key Vault to store their keys or to use the Key Vault extensions to consume the local and cloud cryptographic providers, they will need this package.

Key Vault is designed for high-value master keys, and throttling limits per Key Vault are designed with this in mind. When performing client-side encryption with Key Vault, the preferred model is to use symmetric master keys stored as secrets in Key Vault and cached locally. Users must do the following:

1. Create a secret offline and upload it to Key Vault.
2. Use the secret's base identifier as a parameter to resolve the current version of the secret for encryption and cache this information locally. Use CachingKeyResolver for caching; users are not expected to implement their own caching logic.
3. Use the caching resolver as an input while creating the encryption policy.

More information regarding Key Vault usage can be found in the [encryption code samples](#).

## Best practices

Encryption support is available only in the storage client library for .NET. Windows Phone and Windows Runtime do not currently support encryption.

## IMPORTANT

Be aware of these important points when using client-side encryption:

- When reading from or writing to an encrypted blob, use whole blob upload commands and range/whole blob download commands. Avoid writing to an encrypted blob using protocol operations such as Put Block, Put Block List, Write Pages, Clear Pages, or Append Block; otherwise you may corrupt the encrypted blob and make it unreadable.
- For tables, a similar constraint exists. Be careful to not update encrypted properties without updating the encryption metadata.
- If you set metadata on the encrypted blob, you may overwrite the encryption-related metadata required for decryption, since setting metadata is not additive. This is also true for snapshots; avoid specifying metadata while creating a snapshot of an encrypted blob. If metadata must be set, be sure to call the **FetchAttributes** method first to get the current encryption metadata, and avoid concurrent writes while metadata is being set.
- Enable the **RequireEncryption** property in the default request options for users that should work only with encrypted data. See below for more info.

## Client API / Interface

While creating an **EncryptionPolicy** object, users can provide only a Key (implementing **IKey**), only a resolver (implementing **IKeyResolver**), or both. **IKey** is the basic key type that is identified using a key identifier and that provides the logic for wrapping/unwrapping. **IKeyResolver** is used to resolve a key during the decryption process. It defines a **ResolveKey** method that returns an **IKey** given a key identifier. This provides users the ability to choose between multiple keys that are managed in multiple locations.

- For encryption, the key is used always and the absence of a key will result in an error.
- For decryption:
  - The key resolver is invoked if specified to get the key. If the resolver is specified but does not have a mapping for the key identifier, an error is thrown.
  - If resolver is not specified but a key is specified, the key is used if its identifier matches the required key identifier. If the identifier does not match, an error is thrown.

The code examples in this article demonstrate setting an encryption policy and working with encrypted data, but do not demonstrate working with Azure Key Vault. The [encryption samples](#) on GitHub demonstrate a more detailed end-to-end scenario for blobs, queues and tables, along with Key Vault integration.

### RequireEncryption mode

Users can optionally enable a mode of operation where all uploads and downloads must be encrypted. In this mode, attempts to upload data without an encryption policy or download data that is not encrypted on the service will fail on the client. The **RequireEncryption** property of the request options object controls this behavior. If your application will encrypt all objects stored in Azure Storage, then you can set the **RequireEncryption** property on the default request options for the service client object. For example, set

`CloudBlobClient.DefaultRequestOptions.RequireEncryption` to `true` to require encryption for all blob operations performed through that client object.

### Blob service encryption

Create a **BlobEncryptionPolicy** object and set it in the request options (per API or at a client level by using **DefaultRequestOptions**). Everything else will be handled by the client library internally.

```

// Create the IKey used for encryption.
RsaKey key = new RsaKey("private:key1" /* key identifier */);

// Create the encryption policy to be used for upload and download.
BlobEncryptionPolicy policy = new BlobEncryptionPolicy(key, null);

// Set the encryption policy on the request options.
BlobRequestOptions options = new BlobRequestOptions() { EncryptionPolicy = policy };

// Upload the encrypted contents to the blob.
blob.UploadFromStream(stream, size, null, options, null);

// Download and decrypt the encrypted contents from the blob.
MemoryStream outputStream = new MemoryStream();
blob.DownloadToStream(outputStream, null, options, null);

```

## Queue service encryption

Create a `QueueEncryptionPolicy` object and set it in the request options (per API or at a client level by using `DefaultRequestOptions`). Everything else will be handled by the client library internally.

```

// Create the IKey used for encryption.
RsaKey key = new RsaKey("private:key1" /* key identifier */);

// Create the encryption policy to be used for upload and download.
QueueEncryptionPolicy policy = new QueueEncryptionPolicy(key, null);

// Add message
QueueRequestOptions options = new QueueRequestOptions() { EncryptionPolicy = policy };
queue.AddMessage(message, null, null, options, null);

// Retrieve message
CloudQueueMessage retrMessage = queue.GetMessage(null, options, null);

```

## Table service encryption

In addition to creating an encryption policy and setting it on request options, you must either specify an `EncryptionResolver` in `TableRequestOptions`, or set the `[EncryptProperty]` attribute on the entity.

### Using the resolver

```

// Create the IKey used for encryption.
RsaKey key = new RsaKey("private:key1" /* key identifier */);

// Create the encryption policy to be used for upload and download.
TableEncryptionPolicy policy = new TableEncryptionPolicy(key, null);

TableRequestOptions options = new TableRequestOptions()
{
 EncryptionResolver = (pk, rk, propName) =>
 {
 if (propName == "foo")
 {
 return true;
 }
 return false;
 },
 EncryptionPolicy = policy
};

// Insert Entity
currentTable.Execute(TableOperation.Insert(ent), options, null);

// Retrieve Entity
// No need to specify an encryption resolver for retrieve
TableRequestOptions retrieveOptions = new TableRequestOptions()
{
 EncryptionPolicy = policy
};

TableOperation operation = TableOperation.Retrieve(ent.PartitionKey, ent.RowKey);
TableResult result = currentTable.Execute(operation, retrieveOptions, null);

```

#### Using attributes

As mentioned above, if the entity implements `TableEntity`, then the properties can be decorated with the `[EncryptProperty]` attribute instead of specifying the `EncryptionResolver`.

```

[EncryptProperty]
public string EncryptedProperty1 { get; set; }

```

## Encryption and performance

Note that encrypting your storage data results in additional performance overhead. The content key and IV must be generated, the content itself must be encrypted, and additional meta-data must be formatted and uploaded. This overhead will vary depending on the quantity of data being encrypted. We recommend that customers always test their applications for performance during development.

## Next steps

- [Tutorial: Encrypt and decrypt blobs in Microsoft Azure Storage using Azure Key Vault](#)
- Download the [Azure Storage Client Library for .NET NuGet package](#)
- Download the Azure Key Vault NuGet [Core](#), [Client](#), and [Extensions](#) packages
- Visit the [Azure Key Vault Documentation](#)

# Client-Side Encryption and Azure Key Vault with Java for Microsoft Azure Storage

4/16/2020 • 13 minutes to read • [Edit Online](#)

## Overview

The [Azure Storage Client Library for Java](#) 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.

## Encryption and decryption via the envelope technique

The processes of encryption and decryption follow the envelope technique.

### Encryption via the envelope technique

Encryption via the envelope technique works in the following way:

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 this 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 an asymmetric key pair or a symmetric key and can be managed locally or stored in Azure Key Vaults.  
The 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 the Azure Storage service. The wrapped key along with some additional encryption metadata is either stored as metadata (on a blob) or interpolated with the encrypted data (queue messages and table entities).

### Decryption via the envelope technique

Decryption via the envelope technique works in the following way:

1. The client library assumes that the user is managing the key encryption key (KEK) either locally or in Azure Key Vaults. The user does not need to know the specific key that was used for encryption. Instead, a key resolver which 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 on the service.
3. The wrapped content encryption key (CEK) is then unwrapped (decrypted) using the key encryption key (KEK). Here again, the client library does not have access to KEK. It simply invokes the custom or Key Vault provider's unwrapping algorithm.
4. The content encryption key (CEK) is then used to decrypt the encrypted user data.

## Encryption Mechanism

The storage client library uses [AES](#) in order to encrypt user data. Specifically, [Cipher Block Chaining \(CBC\)](#) mode with AES. Each service works somewhat differently, so we will discuss each of them here.

### Blobs

The client library currently supports encryption of whole blobs only. Specifically, encryption is supported when users use the `upload*` methods or the `openOutputStream` method. For downloads, both complete and range

downloads are supported.

During encryption, the client library will generate a random Initialization Vector (IV) of 16 bytes, together with a random content encryption key (CEK) of 32 bytes, and perform 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 on the service.

#### WARNING

If you are editing or uploading your own metadata for the blob, you need to ensure that this metadata is preserved. If you upload new metadata without this metadata, the wrapped CEK, IV and other metadata will be lost and the blob content will never be retrievable again.

Downloading an encrypted blob involves retrieving the content of the entire blob using the **download/openInputStream** convenience methods. The wrapped CEK is unwrapped and used together with the IV (stored as blob metadata in this case) to return the decrypted data to the users.

Downloading an arbitrary range (**downloadRange** methods) 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.

#### Queues

Since queue messages can be of any format, the client library defines a custom format that includes the Initialization Vector (IV) and the encrypted content encryption key (CEK) in the message text.

During encryption, the client library generates a random IV of 16 bytes along with a random CEK of 32 bytes and performs envelope encryption of the queue message text using this information. The wrapped CEK and some additional encryption metadata are then added to the encrypted queue message. This modified message (shown below) is stored on the service.

```
<MessageText>
{"EncryptedMessageContents": "6k0u8Rq1C3+M1Q04a1KLmWthWXSmHV3mEfxBAgP9QGTU++MKn2uPq3t2UjF1D06w", "EncryptionData"
 : {...}}</MessageText>
```

During decryption, the wrapped key is extracted from the queue message and unwrapped. The IV is also extracted from the queue message and used along with the unwrapped key to decrypt the queue message data. Note that the encryption metadata is small (under 500 bytes), so while it does count toward the 64KB limit for a queue message, the impact should be manageable.

#### Tables

The client library supports encryption of entity properties for insert and replace operations.

#### NOTE

Merge is not currently supported. Since a subset of properties may have been encrypted previously using a different key, simply merging the new properties and updating the metadata will result in data loss. Merging either requires making extra service calls to read the pre-existing entity from the service, or using a new key per property, both of which are not suitable for performance reasons.

Table data encryption works as follows:

1. Users specify the properties to be encrypted.

2. The client library generates a random Initialization Vector (IV) of 16 bytes along with a random content encryption key (CEK) of 32 bytes for every entity, and performs envelope encryption on the individual properties to be encrypted by deriving a new IV per property. The encrypted property is stored as binary data.
3. The wrapped CEK and some additional encryption metadata are then stored as two additional reserved properties. The first reserved property (`_ClientEncryptionMetadata1`) is a string property that holds the information about IV, version, and wrapped key. The second reserved property (`_ClientEncryptionMetadata2`) is a binary property that holds the information about the properties that are encrypted. The information in this second property (`_ClientEncryptionMetadata2`) is itself encrypted.
4. Due to these additional reserved properties required for encryption, users may now have only 250 custom properties instead of 252. The total size of the entity must be less than 1MB.

Note that only string properties can be encrypted. If other types of properties are to be encrypted, they must be converted to strings. The encrypted strings are stored on the service as binary properties, and they are converted back to strings after decryption.

For tables, in addition to the encryption policy, users must specify the properties to be encrypted. This can be done by either specifying an [Encrypt] attribute (for POCO entities that derive from `TableEntity`) or an encryption resolver in request options. An encryption resolver is a delegate that takes a partition key, row key, and property name and returns a boolean that indicates whether that property should be encrypted. During encryption, the client library will use this information to decide whether a property should be encrypted while writing to the wire. The delegate also provides for the possibility of logic around how properties are encrypted. (For example, if X, then encrypt property A; otherwise encrypt properties A and B.) Note that it is not necessary to provide this information while reading or querying entities.

## Batch Operations

In batch operations, the same KEK will be used across all the rows in that batch operation because the client library only allows one options object (and hence one policy/KEK) per batch operation. However, the client library will internally generate a new random IV and random CEK per row in the batch. Users can also choose to encrypt different properties for every operation in the batch by defining this behavior in the encryption resolver.

## Queries

### NOTE

Because the entities are encrypted, you cannot run queries that filter on an encrypted property. If you try, results will be incorrect, because the service would be trying to compare encrypted data with unencrypted data.

To perform query operations, you must specify a key resolver that is able to resolve all the keys in the result set. If an entity contained in the query result cannot be resolved to a provider, the client library will throw an error. For any query that performs server side projections, the client library will add the special encryption metadata properties (`_ClientEncryptionMetadata1` and `_ClientEncryptionMetadata2`) by default to the selected columns.

## Azure Key Vault

Azure Key Vault helps safeguard cryptographic keys and secrets used by cloud applications and services. By using Azure Key Vault, users can encrypt keys and secrets (such as authentication keys, storage account keys, data encryption keys, .PFX files, and passwords) by using keys that are protected by hardware security modules (HSMs). For more information, see [What is Azure Key Vault?](#).

The storage client library uses the Key Vault core library in order to provide a common framework across Azure for managing keys. Users also get the additional benefit of using the Key Vault extensions library. The extensions library provides useful functionality around simple and seamless Symmetric/RSA local and cloud key providers as well as with aggregation and caching.

## Interface and dependencies

There are three Key Vault packages:

- azure-keyvault-core contains the IKey and IKeyResolver. It is a small package with no dependencies. The storage client library for Java defines it as a dependency.
- azure-keyvault contains the Key Vault REST client.
- azure-keyvault-extensions contains extension code that includes implementations of cryptographic algorithms and an RSAKey and a SymmetricKey. It depends on the Core and KeyVault namespaces and provides functionality to define an aggregate resolver (when users want to use multiple key providers) and a caching key resolver. Although the storage client library does not directly depend on this package, if users wish to use Azure Key Vault to store their keys or to use the Key Vault extensions to consume the local and cloud cryptographic providers, they will need this package.

Key Vault is designed for high-value master keys, and throttling limits per Key Vault are designed with this in mind. When performing client-side encryption with Key Vault, the preferred model is to use symmetric master keys stored as secrets in Key Vault and cached locally. Users must do the following:

1. Create a secret offline and upload it to Key Vault.
2. Use the secret's base identifier as a parameter to resolve the current version of the secret for encryption and cache this information locally. Use CachingKeyResolver for caching; users are not expected to implement their own caching logic.
3. Use the caching resolver as an input while creating the encryption policy. More information regarding Key Vault usage can be found in the encryption code samples.

## Best practices

Encryption support is available only in the storage client library for Java.

### IMPORTANT

Be aware of these important points when using client-side encryption:

- When reading from or writing to an encrypted blob, use whole blob upload commands and range/whole blob download commands. Avoid writing to an encrypted blob using protocol operations such as Put Block, Put Block List, Write Pages, Clear Pages, or Append Block; otherwise you may corrupt the encrypted blob and make it unreadable.
- For tables, a similar constraint exists. Be careful to not update encrypted properties without updating the encryption metadata.
- If you set metadata on the encrypted blob, you may overwrite the encryption-related metadata required for decryption, since setting metadata is not additive. This is also true for snapshots; avoid specifying metadata while creating a snapshot of an encrypted blob. If metadata must be set, be sure to call the `downloadAttributes` method first to get the current encryption metadata, and avoid concurrent writes while metadata is being set.
- Enable the `requireEncryption` flag in the default request options for users that should work only with encrypted data. See below for more info.

## Client API / Interface

While creating an EncryptionPolicy object, users can provide only a Key (implementing IKey), only a resolver (implementing IKeyResolver), or both. IKey is the basic key type that is identified using a key identifier and that provides the logic for wrapping/unwrapping. IKeyResolver is used to resolve a key during the decryption process. It defines a ResolveKey method that returns an IKey given a key identifier. This provides users the ability to choose between multiple keys that are managed in multiple locations.

- For encryption, the key is used always and the absence of a key will result in an error.

- For decryption:
  - The key resolver is invoked if specified to get the key. If the resolver is specified but does not have a mapping for the key identifier, an error is thrown.
  - If resolver is not specified but a key is specified, the key is used if its identifier matches the required key identifier. If the identifier does not match, an error is thrown.

The [encryption samples](#) demonstrate a more detailed end-to-end scenario for blobs, queues and tables, along with Key Vault integration.

## RequireEncryption mode

Users can optionally enable a mode of operation where all uploads and downloads must be encrypted. In this mode, attempts to upload data without an encryption policy or download data that is not encrypted on the service will fail on the client. The **requireEncryption** flag of the request options object controls this behavior. If your application will encrypt all objects stored in Azure Storage, then you can set the **requireEncryption** property on the default request options for the service client object.

For example, use `CloudBlobClient.getDefaultRequestOptions().setRequireEncryption(true)` to require encryption for all blob operations performed through that client object.

## Blob service encryption

Create a **BlobEncryptionPolicy** object and set it in the request options (per API or at a client level by using **DefaultRequestOptions**). Everything else will be handled by the client library internally.

```
// Create the IKey used for encryption.
RsaKey key = new RsaKey("private:key1" /* key identifier */);

// Create the encryption policy to be used for upload and download.
BlobEncryptionPolicy policy = new BlobEncryptionPolicy(key, null);

// Set the encryption policy on the request options.
BlobRequestOptions options = new BlobRequestOptions();
options.setEncryptionPolicy(policy);

// Upload the encrypted contents to the blob.
blob.upload(stream, size, null, options, null);

// Download and decrypt the encrypted contents from the blob.
ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
blob.download(outputStream, null, options, null);
```

## Queue service encryption

Create a **QueueEncryptionPolicy** object and set it in the request options (per API or at a client level by using **DefaultRequestOptions**). Everything else will be handled by the client library internally.

```

// Create the IKey used for encryption.
RsaKey key = new RsaKey("private:key1" /* key identifier */);

// Create the encryption policy to be used for upload and download.
QueueEncryptionPolicy policy = new QueueEncryptionPolicy(key, null);

// Add message
QueueRequestOptions options = new QueueRequestOptions();
options.setEncryptionPolicy(policy);

queue.addMessage(message, 0, 0, options, null);

// Retrieve message
CloudQueueMessage retrMessage = queue.retrieveMessage(30, options, null);

```

## Table service encryption

In addition to creating an encryption policy and setting it on request options, you must either specify an **EncryptionResolver** in **TableRequestOptions**, or set the [Encrypt] attribute on the entity's getter and setter.

## Using the resolver

```

// Create the IKey used for encryption.
RsaKey key = new RsaKey("private:key1" /* key identifier */);

// Create the encryption policy to be used for upload and download.
TableEncryptionPolicy policy = new TableEncryptionPolicy(key, null);

TableRequestOptions options = new TableRequestOptions()
options.setEncryptionPolicy(policy);
options.setEncryptionResolver(new EncryptionResolver() {
 public boolean encryptionResolver(String pk, String rk, String key) {
 if (key == "foo")
 {
 return true;
 }
 return false;
 }
});

// Insert Entity
currentTable.execute(TableOperation.insert(ent), options, null);

// Retrieve Entity
// No need to specify an encryption resolver for retrieve
TableRequestOptions retrieveOptions = new TableRequestOptions()
retrieveOptions.setEncryptionPolicy(policy);

TableOperation operation = TableOperation.retrieve(ent.PartitionKey, ent.RowKey, DynamicTableEntity.class);
TableResult result = currentTable.execute(operation, retrieveOptions, null);

```

## Using attributes

As mentioned above, if the entity implements **TableEntity**, then the properties getter and setter can be decorated with the [Encrypt] attribute instead of specifying the **EncryptionResolver**.

```
private string encryptedProperty1;

@Encrypt
public String getEncryptedProperty1 () {
 return this.encryptedProperty1;
}

@Encrypt
public void setEncryptedProperty1(final String encryptedProperty1) {
 this.encryptedProperty1 = encryptedProperty1;
}
```

## Encryption and performance

Note that encrypting your storage data results in additional performance overhead. The content key and IV must be generated, the content itself must be encrypted, and additional meta-data must be formatted and uploaded. This overhead will vary depending on the quantity of data being encrypted. We recommend that customers always test their applications for performance during development.

## Next steps

- Download the [Azure Storage Client Library for Java Maven package](#)
- Download the [Azure Storage Client Library for Java Source Code from GitHub](#)
- Download the Azure Key Vault Maven Library for Java Maven packages:
  - [Core](#) package
  - [Client](#) package
- Visit the [Azure Key Vault Documentation](#)

# Client-side encryption with Python

12/5/2019 • 13 minutes to read • [Edit Online](#)

## Overview

The [Azure Storage Client Library for Python](#) supports encrypting data within client applications before uploading to Azure Storage, and decrypting data while downloading to the client.

### NOTE

The Azure Storage Python library is in preview.

## Encryption and decryption via the envelope technique

The processes of encryption and decryption follow the envelope technique.

### Encryption via the envelope technique

Encryption via the envelope technique works in the following way:

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 this 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 an asymmetric key pair or a symmetric key, which is managed locally. The storage client library itself never has access to KEK. The library invokes the key wrapping algorithm that is provided by the KEK. Users can choose to use custom providers for key wrapping/unwrapping if desired.
4. The encrypted data is then uploaded to the Azure Storage service. The wrapped key along with some additional encryption metadata is either stored as metadata (on a blob) or interpolated with the encrypted data (queue messages and table entities).

### Decryption via the envelope technique

Decryption via the envelope technique works in the following way:

1. The client library assumes that the user is managing the key encryption key (KEK) locally. The user does not need to know the specific key that was used for encryption. Instead, a key resolver, which 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 on the service.
3. The wrapped content encryption key (CEK) is then unwrapped (decrypted) using the key encryption key (KEK). Here again, the client library does not have access to KEK. It simply invokes the custom provider's unwrapping algorithm.
4. The content encryption key (CEK) is then used to decrypt the encrypted user data.

## Encryption Mechanism

The storage client library uses [AES](#) in order to encrypt user data. Specifically, [Cipher Block Chaining \(CBC\)](#) mode with AES. Each service works somewhat differently, so we will discuss each of them here.

### Blobs

The client library currently supports encryption of whole blobs only. Specifically, encryption is supported when

users use the `create*` methods. For downloads, both complete and range downloads are supported, and parallelization of both upload and download is available.

During encryption, the client library will generate a random Initialization Vector (IV) of 16 bytes, together with a random content encryption key (CEK) of 32 bytes, and perform 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 on the service.

#### WARNING

If you are editing or uploading your own metadata for the blob, you need to ensure that this metadata is preserved. If you upload new metadata without this metadata, the wrapped CEK, IV and other metadata will be lost and the blob content will never be retrievable again.

Downloading an encrypted blob involves retrieving the content of the entire blob using the `get*` convenience methods. The wrapped CEK is unwrapped and used together with the IV (stored as blob metadata in this case) to return the decrypted data to the users.

Downloading an arbitrary range (`get*` methods with range parameters passed in) 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.

Block blobs and page blobs only can be encrypted/decrypted using this scheme. There is currently no support for encrypting append blobs.

#### Queues

Since queue messages can be of any format, the client library defines a custom format that includes the Initialization Vector (IV) and the encrypted content encryption key (CEK) in the message text.

During encryption, the client library generates a random IV of 16 bytes along with a random CEK of 32 bytes and performs envelope encryption of the queue message text using this information. The wrapped CEK and some additional encryption metadata are then added to the encrypted queue message. This modified message (shown below) is stored on the service.

```
<MessageText>
{"EncryptedMessageContents": "6k0u8Rq1C3+M1Q04a1KLmWthWXSmHV3mEfxBAgP9QGTU++MKn2uPq3t2UjF1D06w", "EncryptionData"
 : {...}}</MessageText>
```

During decryption, the wrapped key is extracted from the queue message and unwrapped. The IV is also extracted from the queue message and used along with the unwrapped key to decrypt the queue message data. Note that the encryption metadata is small (under 500 bytes), so while it does count toward the 64KB limit for a queue message, the impact should be manageable.

#### Tables

The client library supports encryption of entity properties for insert and replace operations.

#### NOTE

Merge is not currently supported. Since a subset of properties may have been encrypted previously using a different key, simply merging the new properties and updating the metadata will result in data loss. Merging either requires making extra service calls to read the pre-existing entity from the service, or using a new key per property, both of which are not suitable for performance reasons.

Table data encryption works as follows:

1. Users specify the properties to be encrypted.
2. The client library generates a random Initialization Vector (IV) of 16 bytes along with a random content encryption key (CEK) of 32 bytes for every entity, and performs envelope encryption on the individual properties to be encrypted by deriving a new IV per property. The encrypted property is stored as binary data.
3. The wrapped CEK and some additional encryption metadata are then stored as two additional reserved properties. The first reserved property (`_ClientEncryptionMetadata1`) is a string property that holds the information about IV, version, and wrapped key. The second reserved property (`_ClientEncryptionMetadata2`) is a binary property that holds the information about the properties that are encrypted. The information in this second property (`_ClientEncryptionMetadata2`) is itself encrypted.
4. Due to these additional reserved properties required for encryption, users may now have only 250 custom properties instead of 252. The total size of the entity must be less than 1MB.

Note that only string properties can be encrypted. If other types of properties are to be encrypted, they must be converted to strings. The encrypted strings are stored on the service as binary properties, and they are converted back to strings (raw strings, not EntityProperties with type `EdmType.STRING`) after decryption.

For tables, in addition to the encryption policy, users must specify the properties to be encrypted. This can be done by either storing these properties in `TableEntity` objects with the type set to `EdmType.STRING` and `encrypt` set to true or setting the `encryption_resolver_function` on the `tableservice` object. An encryption resolver is a function that takes a partition key, row key, and property name and returns a boolean that indicates whether that property should be encrypted. During encryption, the client library will use this information to decide whether a property should be encrypted while writing to the wire. The delegate also provides for the possibility of logic around how properties are encrypted. (For example, if X, then encrypt property A; otherwise encrypt properties A and B.) Note that it is not necessary to provide this information while reading or querying entities.

## Batch Operations

One encryption policy applies to all rows in the batch. The client library will internally generate a new random IV and random CEK per row in the batch. Users can also choose to encrypt different properties for every operation in the batch by defining this behavior in the encryption resolver. If a batch is created as a context manager through the `tableservice batch()` method, the `tableservice`'s encryption policy will automatically be applied to the batch. If a batch is created explicitly by calling the constructor, the encryption policy must be passed as a parameter and left unmodified for the lifetime of the batch. Note that entities are encrypted as they are inserted into the batch using the batch's encryption policy (entities are NOT encrypted at the time of committing the batch using the `tableservice`'s encryption policy).

## Queries

### NOTE

Because the entities are encrypted, you cannot run queries that filter on an encrypted property. If you try, results will be incorrect, because the service would be trying to compare encrypted data with unencrypted data.

To perform query operations, you must specify a key resolver that is able to resolve all the keys in the result set. If an entity contained in the query result cannot be resolved to a provider, the client library will throw an error. For any query that performs server side projections, the client library will add the special encryption metadata properties (`_ClientEncryptionMetadata1` and `_ClientEncryptionMetadata2`) by default to the selected columns.

## IMPORTANT

Be aware of these important points when using client-side encryption:

- When reading from or writing to an encrypted blob, use whole blob upload commands and range/whole blob download commands. Avoid writing to an encrypted blob using protocol operations such as Put Block, Put Block List, Write Pages, or Clear Pages; otherwise you may corrupt the encrypted blob and make it unreadable.
- For tables, a similar constraint exists. Be careful to not update encrypted properties without updating the encryption metadata.
- If you set metadata on the encrypted blob, you may overwrite the encryption-related metadata required for decryption, since setting metadata is not additive. This is also true for snapshots; avoid specifying metadata while creating a snapshot of an encrypted blob. If metadata must be set, be sure to call the `get_blob_metadata` method first to get the current encryption metadata, and avoid concurrent writes while metadata is being set.
- Enable the `require_encryption` flag on the service object for users that should work only with encrypted data. See below for more info.

The storage client library expects the provided KEK and key resolver to implement the following interface. [Azure Key Vault](#) support for Python KEK management is pending and will be integrated into this library when completed.

## Client API / Interface

After a storage service object (i.e. `blockblobservice`) has been created, the user may assign values to the fields that constitute an encryption policy: `key_encryption_key`, `key_resolver_function`, and `require_encryption`. Users can provide only a KEK, only a resolver, or both. `key_encryption_key` is the basic key type that is identified using a key identifier and that provides the logic for wrapping/unwrapping. `key_resolver_function` is used to resolve a key during the decryption process. It returns a valid KEK given a key identifier. This provides users the ability to choose between multiple keys that are managed in multiple locations.

The KEK must implement the following methods to successfully encrypt data:

- `wrap_key(cek)`: Wraps the specified CEK (bytes) using an algorithm of the user's choice. Returns the wrapped key.
- `get_key_wrap_algorithm()`: Returns the algorithm used to wrap keys.
- `get_kid()`: Returns the string key id for this KEK. The KEK must implement the following methods to successfully decrypt data:
- `unwrap_key(cek, algorithm)`: Returns the unwrapped form of the specified CEK using the string-specified algorithm.
- `get_kid()`: Returns a string key id for this KEK.

The key resolver must at least implement a method that, given a key id, returns the corresponding KEK implementing the interface above. Only this method is to be assigned to the `key_resolver_function` property on the service object.

- For encryption, the key is used always and the absence of a key will result in an error.
- For decryption:
  - The key resolver is invoked if specified to get the key. If the resolver is specified but does not have a mapping for the key identifier, an error is thrown.
  - If resolver is not specified but a key is specified, the key is used if its identifier matches the required key identifier. If the identifier does not match, an error is thrown.

The encryption samples in `azure.storage.samples` demonstrate a more detailed end-to-end scenario for blobs, queues and tables. Sample implementations of the KEK and key resolver are provided in the sample files as `KeyWrapper` and `KeyResolver` respectively.

## RequireEncryption mode

Users can optionally enable a mode of operation where all uploads and downloads must be encrypted. In this mode, attempts to upload data without an encryption policy or download data that is not encrypted on the service will fail on the client. The `require_encryption` flag on the service object controls this behavior.

## Blob service encryption

Set the encryption policy fields on the `blockblobservice` object. Everything else will be handled by the client library internally.

```
Create the KEK used for encryption.
KeyWrapper is the provided sample implementation, but the user may use their own object as long as it
implements the interface above.
kek = KeyWrapper('local:key1') # Key identifier

Create the key resolver used for decryption.
KeyResolver is the provided sample implementation, but the user may use whatever implementation they choose
so long as the function set on the service object behaves appropriately.
key_resolver = KeyResolver()
key_resolver.put_key(kek)

Set the KEK and key resolver on the service object.
my_block_blob_service.key_encryption_key = kek
my_block_blob_service.key_resolver_funcion = key_resolver.resolve_key

Upload the encrypted contents to the blob.
my_block_blob_service.create_blob_from_stream(
 container_name, blob_name, stream)

Download and decrypt the encrypted contents from the blob.
blob = my_block_blob_service.get_blob_to_bytes(container_name, blob_name)
```

## Queue service encryption

Set the encryption policy fields on the `queueservice` object. Everything else will be handled by the client library internally.

```
Create the KEK used for encryption.
KeyWrapper is the provided sample implementation, but the user may use their own object as long as it
implements the interface above.
kek = KeyWrapper('local:key1') # Key identifier

Create the key resolver used for decryption.
KeyResolver is the provided sample implementation, but the user may use whatever implementation they choose
so long as the function set on the service object behaves appropriately.
key_resolver = KeyResolver()
key_resolver.put_key(kek)

Set the KEK and key resolver on the service object.
my_queue_service.key_encryption_key = kek
my_queue_service.key_resolver_funcion = key_resolver.resolve_key

Add message
my_queue_service.put_message(queue_name, content)

Retrieve message
retrieved_message_list = my_queue_service.get_messages(queue_name)
```

## Table service encryption

In addition to creating an encryption policy and setting it on request options, you must either specify an `encryption_resolver_function` on the `tableservice`, or set the `encrypt` attribute on the `EntityProperty`.

## Using the resolver

```
Create the KEK used for encryption.
KeyWrapper is the provided sample implementation, but the user may use their own object as long as it
implements the interface above.
kek = KeyWrapper('local:key1') # Key identifier

Create the key resolver used for decryption.
KeyResolver is the provided sample implementation, but the user may use whatever implementation they choose
so long as the function set on the service object behaves appropriately.
key_resolver = KeyResolver()
key_resolver.put_key(kek)

Define the encryption resolver_function.

def my_encryption_resolver(pk, rk, property_name):
 if property_name == 'foo':
 return True
 return False

Set the KEK and key resolver on the service object.
my_table_service.key_encryption_key = kek
my_table_service.key_resolver_funcion = key_resolver.resolve_key
my_table_service.encryption_resolver_function = my_encryption_resolver

Insert Entity
my_table_service.insert_entity(table_name, entity)

Retrieve Entity
Note: No need to specify an encryption resolver for retrieve, but it is harmless to leave the property set.
my_table_service.get_entity(
 table_name, entity['PartitionKey'], entity['RowKey'])
```

## Using attributes

As mentioned above, a property may be marked for encryption by storing it in an EntityProperty object and setting the encrypt field.

```
encrypted_property_1 = EntityProperty(EdmType.STRING, value, encrypt=True)
```

## Encryption and performance

Note that encrypting your storage data results in additional performance overhead. The content key and IV must be generated, the content itself must be encrypted, and additional metadata must be formatted and uploaded. This overhead will vary depending on the quantity of data being encrypted. We recommend that customers always test their applications for performance during development.

## Next steps

- Download the [Azure Storage Client Library for Java PyPi package](#)
- Download the [Azure Storage Client Library for Python Source Code from GitHub](#)

# Require secure transfer to ensure secure connections

4/22/2020 • 2 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.

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, SMB 3.0 without encryption, or some versions of the Linux SMB client.

By default, the **Secure transfer required** property is enabled when you create a storage account.

## 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. And classic storage accounts are not supported.

## 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. Under **Secure transfer required**, select **Enabled**.

**Create storage account**

Basics Advanced Tags Review + create

**SECURITY**

Secure transfer required Enabled

**VIRTUAL NETWORKS**

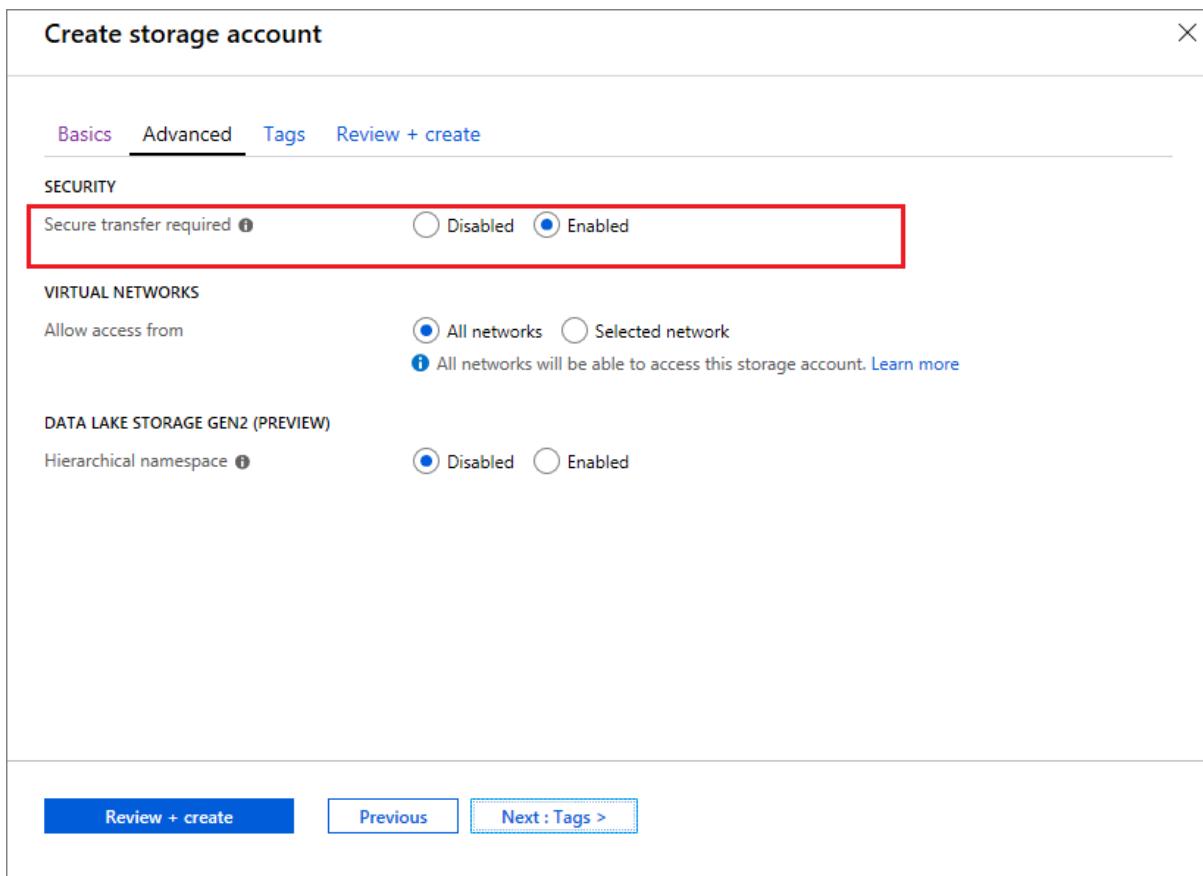
Allow access from All networks

All networks will be able to access this storage account. [Learn more](#)

**DATA LAKE STORAGE GEN2 (PREVIEW)**

Hierarchical namespace Disabled

**Review + create** **Previous** **Next : Tags >**



### 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.

**requiresecurefer - Configuration**

Storage account

Search (Ctrl+ /)

Save Discard

The cost of your storage account depends on the usage and the options you choose below.

Learn more

Performance Standard

Secure transfer required Enabled

Replication Read-access geo-redundant storage (RA-GRS)

**SETTINGS**

Access keys

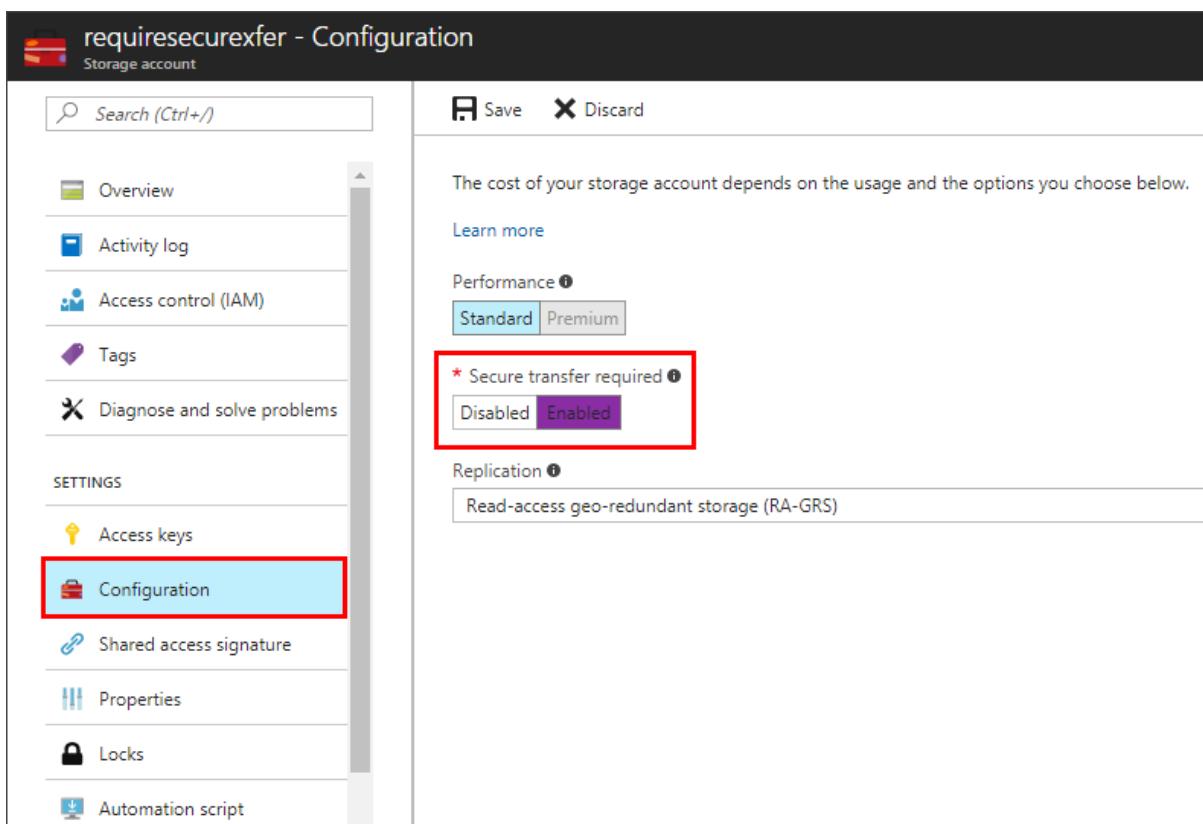
**Configuration** Enabled

Shared access signature

Properties

Locks

Automation script



### Require secure transfer from code

To require secure transfer programmatically, set the *supportsHttpsTrafficOnly* property 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 has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

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 a [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

4/10/2020 • 18 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 used. When network rules are configured, only applications requesting data over the specified set of networks can access a storage account. You can limit access to your storage account to requests originating from specified IP addresses, IP ranges or from a list of subnets in an Azure Virtual Network (VNet).

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 access 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.

## 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 has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

## 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 select 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 to 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 backup and restore VMs by creating an exception. This process is documented in the [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. To limit access to selected networks, you must first change the default action.

### WARNING

Making changes to network rules can impact your applications' ability to connect to Azure Storage. Setting the default network rule to **deny** blocks all access to the data unless specific network rules that **grant** access are also applied. Be sure to grant access to any allowed networks using network rules before you change the default rule to deny access.

### Managing default network access rules

You can manage default network access rules for storage accounts through the Azure portal, PowerShell, or CLIv2.

#### Azure portal

1. Go to the storage account you want to secure.
2. Click on the settings menu called **Firewalls and virtual networks**.
3. To deny access by default, choose to allow access from **Selected networks**. To allow traffic from all networks, choose to allow access from **All networks**.
4. Click **Save** to apply your changes.

#### PowerShell

1. Install the [Azure PowerShell](#) and [sign in](#).
2. Display the status of the default rule for the storage account.

```
(Get-AzStorageAccountNetworkRuleSet -ResourceGroupName "myresourcegroup" -AccountName
"mystorageaccount").DefaultAction
```

3. Set the default rule to deny network access by default.

```
Update-AzStorageAccountNetworkRuleSet -ResourceGroupName "myresourcegroup" -Name "mystorageaccount" -
DefaultAction Deny
```

- Set the default rule to allow network access by default.

```
Update-AzStorageAccountNetworkRuleSet -ResourceGroupName "myresourcegroup" -Name "mystorageaccount" -DefaultAction Allow
```

#### CLIV2

- Install the [Azure CLI](#) and [sign in](#).
- Display the status of the default rule for the storage account.

```
az storage account show --resource-group "myresourcegroup" --name "mystorageaccount" --query networkRuleSet.defaultAction
```

- Set the default rule to deny network access by default.

```
az storage account update --resource-group "myresourcegroup" --name "mystorageaccount" --default-action Deny
```

- Set the default rule to allow network access by default.

```
az storage account update --resource-group "myresourcegroup" --name "mystorageaccount" --default-action Allow
```

## 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.

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 100 virtual network rules, which may be combined with [IP network rules](#).

### Available virtual network regions

In general, service endpoints work between virtual networks and service instances in the same Azure region. When using service endpoints with Azure Storage, this scope grows to include the [paired region](#). 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.

#### NOTE

Service endpoints don't apply to traffic outside the region of the virtual network and the designated region pair. You can only apply network rules granting access from virtual networks to storage accounts in the primary region of a storage account or in the designated paired region.

## Required permissions

To apply a virtual network rule to a storage account, the user must have the appropriate permissions for the subnets being added. The permission needed is *Join Service to a Subnet* and is included in the *Storage Account Contributor* built-in role. It can also be added to custom role definitions.

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.

## Managing virtual network rules

You can manage virtual network rules for storage accounts through the Azure portal, PowerShell, or CLIV2.

### Azure portal

1. Go to the storage account you want to secure.
2. Click on the settings menu called **Firewalls and virtual networks**.
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**, click **Add existing virtual network**, select **Virtual networks** and **Subnets** options, and then click **Add**. To create a new virtual network and grant it access, click **Add new virtual network**. Provide the information necessary to create the new virtual network, and then click **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.

5. To remove a virtual network or subnet rule, click ... to open the context menu for the virtual network or subnet, and click **Remove**.
6. Click **Save** to apply your changes.

### PowerShell

1. Install the [Azure PowerShell](#) and sign in.

2. List virtual network rules.

```
(Get-AzStorageAccountNetworkRuleSet -ResourceGroupName "myresourcegroup" -AccountName
"mystorageaccount").VirtualNetworkRules
```

3. Enable service endpoint for Azure Storage on an existing virtual network and subnet.

```
Get-AzVirtualNetwork -ResourceGroupName "myresourcegroup" -Name "myvnet" | Set-AzVirtualNetworkSubnetConfig -Name "mysubnet" -AddressPrefix "10.0.0.0/24" -ServiceEndpoint "Microsoft.Storage" | Set-AzVirtualNetwork
```

#### 4. Add a network rule for a virtual network and subnet.

```
$subnet = Get-AzVirtualNetwork -ResourceGroupName "myresourcegroup" -Name "myvnet" | Get-AzVirtualNetworkSubnetConfig -Name "mysubnet"
Add-AzStorageAccountNetworkRule -ResourceGroupName "myresourcegroup" -Name "mystorageaccount" -VirtualNetworkResourceId $subnet.Id
```

#### TIP

To add a network rule for a subnet in a VNet belonging to another Azure AD tenant, use a fully-qualified **VirtualNetworkResourceId** parameter in the form "/subscriptions/subscription-ID/resourceGroups/resourceGroupName/providers/Microsoft.Network/virtualNetworks/vNet-name/subnets/subnet-name".

#### 5. Remove a network rule for a virtual network and subnet.

```
$subnet = Get-AzVirtualNetwork -ResourceGroupName "myresourcegroup" -Name "myvnet" | Get-AzVirtualNetworkSubnetConfig -Name "mysubnet"
Remove-AzStorageAccountNetworkRule -ResourceGroupName "myresourcegroup" -Name "mystorageaccount" -VirtualNetworkResourceId $subnet.Id
```

#### IMPORTANT

Be sure to [set the default rule to deny](#), or network rules have no effect.

#### CLIV2

##### 1. Install the [Azure CLI](#) and sign in.

##### 2. List virtual network rules.

```
az storage account network-rule list --resource-group "myresourcegroup" --account-name "mystorageaccount" --query virtualNetworkRules
```

##### 3. Enable service endpoint for Azure Storage on an existing virtual network and subnet.

```
az network vnet subnet update --resource-group "myresourcegroup" --vnet-name "myvnet" --name "mysubnet" --service-endpoints "Microsoft.Storage"
```

##### 4. Add a network rule for a virtual network and subnet.

```
$subnetid=(az network vnet subnet show --resource-group "myresourcegroup" --vnet-name "myvnet" --name "mysubnet" --query id --output tsv)
az storage account network-rule add --resource-group "myresourcegroup" --account-name "mystorageaccount" --subnet $subnetid
```

**TIP**

To add a rule for a subnet in a VNet belonging to another Azure AD tenant, use a fully-qualified subnet ID in the form `"/subscriptions/<subscription-ID>/resourceGroups/<resourceGroupName>/providers/Microsoft.Network/virtualNetworks/<vNet-name>/subnets/<subnet-name>"`.

You can use the **subscription** parameter to retrieve the subnet ID for a VNet belonging to another Azure AD tenant.

## 5. Remove a network rule for a virtual network and subnet.

```
$subnetid=(az network vnet subnet show --resource-group "myresourcegroup" --vnet-name "myvnet" --name "mysubnet" --query id --output tsv)
az storage account network-rule remove --resource-group "myresourcegroup" --account-name "mystorageaccount" --subnet $subnetid
```

**IMPORTANT**

Be sure to [set the default rule to deny](#), or network rules have no effect.

## Grant access from an internet IP range

You can configure storage accounts to allow access from specific public internet IP address ranges. This configuration grants access to specific internet-based services and on-premises networks and blocks general internet traffic.

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`.

**NOTE**

Small address ranges using `/31` or `/32` prefix sizes are not supported. These ranges should be configured using individual IP address rules.

IP network rules are only allowed 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.*`.

**NOTE**

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.

**NOTE**

Services deployed in the same region as the storage account use private Azure IP addresses for communication. Thus, you cannot restrict access to specific Azure services based on their public outbound IP address range.

Only IPV4 addresses are supported for configuration of storage firewall rules.

Each storage account supports up to 100 IP network rules.

### 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.

### Azure portal

1. Go to the storage account you want to secure.
2. Click on the settings menu called **Firewalls and virtual networks**.
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, click the trash can icon next to the address range.
6. Click **Save** to apply your changes.

### PowerShell

1. Install the [Azure PowerShell](#) and [sign in](#).
2. List IP network rules.

```
(Get-AzStorageAccountNetworkRuleSet -ResourceGroupName "myresourcegroup" -AccountName "mystorageaccount").IPRules
```

3. Add a network rule for an individual IP address.

```
Add-AzStorageAccountNetworkRule -ResourceGroupName "myresourcegroup" -AccountName "mystorageaccount" -IPAddressOrRange "16.17.18.19"
```

4. Add a network rule for an IP address range.

```
Add-AzStorageAccountNetworkRule -ResourceGroupName "myresourcegroup" -AccountName "mystorageaccount" -IPAddressOrRange "16.17.18.0/24"
```

5. Remove a network rule for an individual IP address.

```
Remove-AzStorageAccountNetworkRule -ResourceGroupName "myresourcegroup" -AccountName "mystorageaccount" -IPAddressOrRange "16.17.18.19"
```

6. Remove a network rule for an IP address range.

```
Remove-AzStorageAccountNetworkRule -ResourceGroupName "myresourcegroup" -AccountName "mystorageaccount" -IPAddressOrRange "16.17.18.0/24"
```

## IMPORTANT

Be sure to [set the default rule to deny](#), or network rules have no effect.

### CLIv2

1. Install the [Azure CLI](#) and sign in.

2. List IP network rules.

```
az storage account network-rule list --resource-group "myresourcegroup" --account-name
"mystorageaccount" --query ipRules
```

3. Add a network rule for an individual IP address.

```
az storage account network-rule add --resource-group "myresourcegroup" --account-name "mystorageaccount"
--ip-address "16.17.18.19"
```

4. Add a network rule for an IP address range.

```
az storage account network-rule add --resource-group "myresourcegroup" --account-name "mystorageaccount"
--ip-address "16.17.18.0/24"
```

5. Remove a network rule for an individual IP address.

```
az storage account network-rule remove --resource-group "myresourcegroup" --account-name
"mystorageaccount" --ip-address "16.17.18.19"
```

6. Remove a network rule for an IP address range.

```
az storage account network-rule remove --resource-group "myresourcegroup" --account-name
"mystorageaccount" --ip-address "16.17.18.0/24"
```

## IMPORTANT

Be sure to [set the default rule to deny](#), or network rules have no effect.

## Exceptions

Network rules help to create a secure environment for connections between your applications and your data for most scenarios. However, some applications depend on Azure services that cannot be uniquely isolated through virtual network or IP address rules. But such services must be granted to storage to enable full application functionality. In such situations, you can use the *Allow trusted Microsoft services...* setting to enable such services to access your data, logs, or analytics.

### Trusted Microsoft services

Some Microsoft services operate from networks that can't be included in your network rules. You can grant a subset of such trusted Microsoft services access to the storage account, while maintaining network rules for other apps. These trusted services will then use strong authentication to connect to your storage account securely. We've enabled two modes of trusted access for Microsoft services.

- 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.

- Resources of some services can be granted explicit access to your storage account by **assigning an RBAC role** to its system-assigned managed identity.

When you enable the **Allow trusted Microsoft services...** setting, resources of the following services that are registered in the same subscription as your storage account are granted access for a limited set of operations as described:

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> .
Azure Import Export	Microsoft.ImportExport	Enables import of data to Azure and export of data from Azure using Import/Export service. <a href="#">Learn more</a> .
Azure Monitor	Microsoft.Insights	Allows writing of monitoring data to a secured storage account, including resource diagnostic 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. <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> .

The **Allow trusted Microsoft services...** setting also allows a particular instance of the below services to access the storage account, if you explicitly [assign an RBAC role](#) to the [system-assigned managed identity](#) for that resource instance. In this case, the scope of access for the instance corresponds to the RBAC role assigned to the managed identity.

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 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 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 SQL Data Warehouse	Microsoft.Sql	Allows import and export of data from specific SQL Database instances using PolyBase. <a href="#">Learn more</a> .
Azure Stream Analytics	Microsoft.StreamAnalytics	Allows data from a streaming job to be written to Blob storage. This feature is currently in preview. <a href="#">Learn more</a> .
Azure Synapse Analytics	Microsoft.Synapse/workspaces	Enables access to data in Azure Storage from Synapse Analytics.

## Storage analytics data access

In some cases, access to read diagnostic 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. [Learn more about working with storage analytics](#).

## Managing exceptions

You can manage network rule exceptions through the Azure portal, PowerShell, or Azure CLI v2.

### Azure portal

1. Go to the storage account you want to secure.
2. Click on the settings menu called **Firewalls and virtual networks**.
3. Check that you've selected to allow access from **Selected networks**.
4. Under **Exceptions**, select the exceptions you wish to grant.
5. Click **Save** to apply your changes.

## PowerShell

1. Install the [Azure PowerShell](#) and [sign in](#).
2. Display the exceptions for the storage account network rules.

```
(Get-AzStorageAccountNetworkRuleSet -ResourceGroupName "myresourcegroup" -Name "mystorageaccount").Bypass
```

3. Configure the exceptions to the storage account network rules.

```
Update-AzStorageAccountNetworkRuleSet -ResourceGroupName "myresourcegroup" -Name "mystorageaccount" -Bypass AzureServices,Metrics,Logging
```

4. Remove the exceptions to the storage account network rules.

```
Update-AzStorageAccountNetworkRuleSet -ResourceGroupName "myresourcegroup" -Name "mystorageaccount" -Bypass None
```

### IMPORTANT

Be sure to [set the default rule to deny](#), or removing exceptions have no effect.

## CLIV2

1. Install the [Azure CLI](#) and [sign in](#).
2. Display the exceptions for the storage account network rules.

```
az storage account show --resource-group "myresourcegroup" --name "mystorageaccount" --query networkRuleSet.bypass
```

3. Configure the exceptions to the storage account network rules.

```
az storage account update --resource-group "myresourcegroup" --name "mystorageaccount" --bypass Logging Metrics AzureServices
```

4. Remove the exceptions to the storage account network rules.

```
az storage account update --resource-group "myresourcegroup" --name "mystorageaccount" --bypass None
```

### IMPORTANT

Be sure to [set the default rule to deny](#), or removing exceptions have no effect.

## Next steps

Learn more about Azure Network service endpoints in [Service endpoints](#).

Dig deeper into Azure Storage security in [Azure Storage security guide](#).

# Enable secure TLS for Azure Storage client

12/23/2019 • 2 minutes to read • [Edit Online](#)

Transport Layer Security (TLS) and Secure Sockets Layer (SSL) are cryptographic protocols that provide communications security over a computer network. SSL 1.0, 2.0 and 3.0 have been found to be vulnerable. They have been prohibited by RFC. TLS 1.0 becomes insecure for using insecure Block cipher (DES CBC and RC2 CBC) and Stream cipher (RC4). PCI council also suggested the migration to higher TLS versions. For more details, you can see [Transport Layer Security \(TLS\)](#).

Azure Storage has stopped SSL 3.0 since 2015 and uses TLS 1.2 on public HTTPs endpoints but TLS 1.0 and TLS 1.1 are still supported for backward compatibility.

In order to ensure secure and compliant connection to Azure Storage, you need to enable TLS 1.2 or newer version in client side before sending requests to operate Azure Storage service.

## Enable TLS 1.2 in .NET client

For the client to negotiate TLS 1.2, the OS and the .NET Framework version both need to support TLS 1.2. See more details in [Support for TLS 1.2](#).

The following sample shows how to enable TLS 1.2 in your .NET client.

```
static void EnableTls12()
{
 // Enable TLS 1.2 before connecting to Azure Storage
 System.Net.ServicePointManager.SecurityProtocol = System.Net.SecurityProtocolType.Tls12;

 // Connect to Azure Storage
 CloudStorageAccount storageAccount =
 CloudStorageAccount.Parse("DefaultEndpointsProtocol=https;AccountName={yourstorageaccount};AccountKey=
 {yourstorageaccountkey};EndpointSuffix=core.windows.net");
 CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

 CloudBlobContainer container = blobClient.GetContainerReference("foo");
 container.CreateIfNotExists();
}
```

## Enable TLS 1.2 in PowerShell client

### NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

The following sample shows how to enable TLS 1.2 in your PowerShell client.

```

Enable TLS 1.2 before connecting to Azure Storage
[System.Net.ServicePointManager]::SecurityProtocol = [System.Net.SecurityProtocolType]::Tls12;

$resourceGroup = "{YourResourceGroupName}"
$storageAccountName = "{YourStorageAccountName}"
$prefix = "foo"

Connect to Azure Storage
$storageAccount = Get-AzStorageAccount -ResourceGroupName $resourceGroup -Name $storageAccountName
$ctx = $storageAccount.Context
$listOfContainers = Get-AzStorageContainer -Context $ctx -Prefix $prefix
$listOfContainers

```

## Verify TLS 1.2 connection

You can use Fiddler to verify if TLS 1.2 is used actually. Open Fiddler to start capturing client network traffic then execute above sample. Then you can find the TLS version in the connection that the sample makes.

The following screenshot is a sample for the verification.

A screenshot of the Fiddler Web Debugger interface. The main window shows three captured sessions. Session 1 is a 200 HTTP response from config.edge.skygate.com:443. Session 2 is a 200 HTTP response from client-office365-tas.msedge.net:443. Session 3 is a 200 HTTP response from blob.core.windows.net:443. The session details for Session 3 are expanded, showing a yellow status bar at the top stating 'A SSLv3-compatible ClientHello handshake was found. Fiddler extracted the parameters below.' Below this, the 'Version' field is highlighted with a red box and contains the value 'Version: 3.3 (TLS/1.2)'. The 'Ciphers' section lists numerous cipher suites, such as [C02C] TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_GCM\_SHA384, [C02B] TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256, [C030] TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384, [C02F] TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256, [D00F] TLS\_DHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384, [D00E] TLS\_DHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256, [C024] TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA384, [C023] TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA256, [C028] TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA384, [C027] TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256, [C00A] TLS1\_2K\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA, [C009] TLS1\_2K\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA, [C014] TLS1\_2K\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA, [C013] TLS1\_2K\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA, [D009D] TLS\_RSA\_WITH\_AES\_256\_GCM\_SHA384, [D009C] TLS\_RSA\_WITH\_AES\_128\_GCM\_SHA256, and [D003D] TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA256.

## See also

- [Transport Layer Security \(TLS\)](#)
- [PCI compliance on TLS](#)
- [Enable TLS in Java client](#)

# Set and manage immutability policies for Blob storage

3/10/2020 • 4 minutes to read • [Edit Online](#)

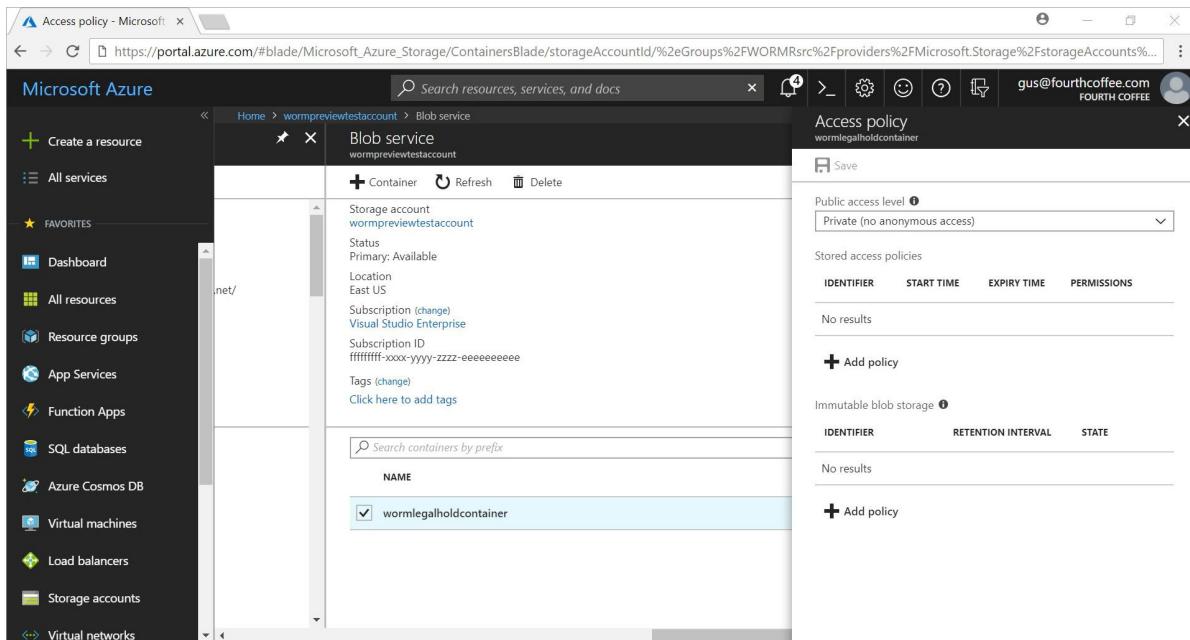
Immutable storage for Azure Blob storage enables users to store business-critical data objects in a WORM (Write Once, Read Many) state. This state makes the data non-erasable and non-modifiable for a user-specified interval. For the duration of the retention interval, blobs can be created and read, but cannot be modified or deleted. Immutable storage is available for general-purpose v2 and Blob storage accounts in all Azure regions.

This article shows how to set and manage immutability policies and legal holds for data in Blob storage using the Azure portal, PowerShell, or Azure CLI. For more information about immutable storage, see [Store business-critical blob data with immutable storage](#).

## Set retention policies and legal holds

- [Portal](#)
- [Azure CLI](#)
- [PowerShell](#)

1. Create a new container or select an existing container to store the blobs that need to be kept in the immutable state. The container must be in a general-purpose v2 or Blob storage account.
2. Select **Access policy** in the container settings. Then select **Add policy** under **Immutable blob storage**.



The screenshot shows the Azure portal interface for managing access policies. The left sidebar shows 'Create a resource' and a list of services like Dashboard, Resource groups, App Services, Function Apps, SQL databases, Azure Cosmos DB, Virtual machines, Load balancers, Storage accounts, and Virtual networks. The main content area is titled 'Access policy' for the container 'wormlegalholdcontainer'. It displays the container's properties: Storage account (wormpreviewtestaccount), Status (Primary: Available), Location (East US), Subscription (Visual Studio Enterprise), Subscription ID (ffffff-xxxx-yyyy-zzzz-eeeeeeee), and Tags. Below this is a search bar for containers by prefix. A table lists the 'Immutable blob storage' policies for the container, showing one row with the identifier 'wormlegalholdcontainer'. At the bottom, there is a 'Save' button and a 'Public access level' dropdown set to 'Private (no anonymous access)'. A large 'Add policy' button is prominently displayed at the bottom right of the policy list.

3. To enable time-based retention, select **Time-based retention** from the drop-down menu.

wormtestcontainer1 - Access policy

Container

Search (Ctrl+ /)

Save

Overview

Access Control (IAM)

SETTINGS

Access policy

Properties

Immutable blob storage

Policy type

- Time-based retention
- Time-based retention
- Legal hold

days

OK Cancel

IDENTIFIER	RETENTION INTERVAL	STATE
No results		

+ Add policy

4. Enter the retention interval in days (acceptable values are 1 to 146000 days).

wormtestcontainer1 - Access policy

Container

Search (Ctrl+ /)

Save

Overview

Access Control (IAM)

SETTINGS

Access policy

Properties

Immutable blob storage

Public access level

Private (no anonymous access)

Stored access policies

IDENTIFIER	START TIME	EXPIRY TIME	PERMISSIONS
Immutable blob storage			

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.

\* Update retention period to

1 days

OK Cancel

VAL	EXPIRY TIME	PERMISSIONS	STATE
			Unlocked

The initial state of the policy is unlocked allowing you to test the feature and make changes to the policy before you lock it. Locking the policy is essential for compliance with regulations like SEC 17a-4.

5. Lock the policy. Right-click the ellipsis (...), and the following menu appears with additional actions:

The screenshot shows the 'wormtestcontainer1 - Access policy' page. In the 'Immutable blob storage' section, there is a table with one row: 'Time-based retention' with a 'RETENTION INTERVAL' of '10 days'. A context menu is open over this row, with the 'Lock policy' option highlighted by a red border.

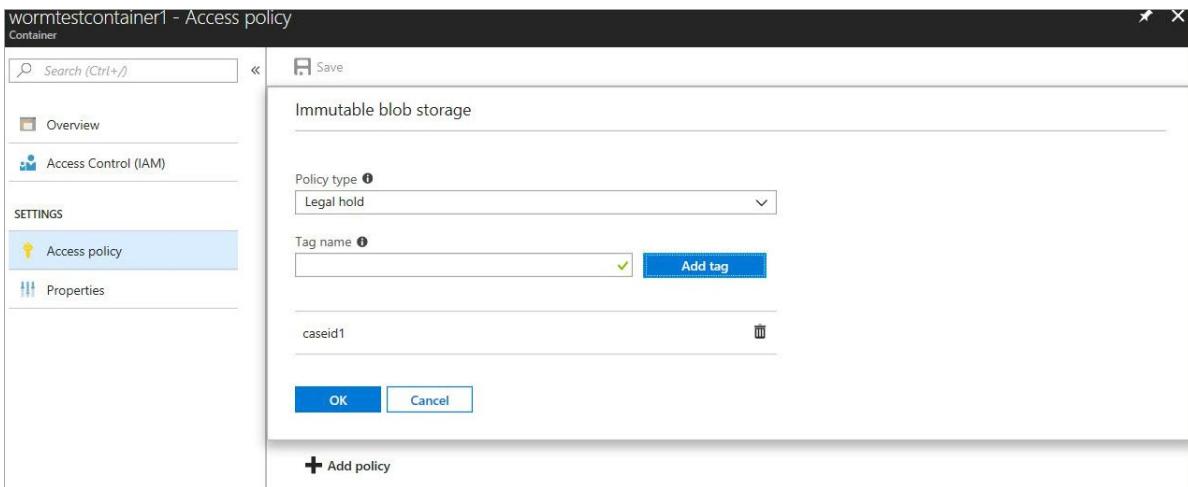
6. Select **Lock Policy** and confirm the lock. The policy is now locked and cannot be deleted, only extensions of the retention interval will be allowed. Blob deletes and overrides are not permitted.

The screenshot shows the 'wormtestcontainer1 - Access policy' page. A confirmation dialog box is overlaid on the screen, titled 'Complete time-based retention policy'. It contains a warning message: 'Completing the retention policy process is irreversible. Ensure the retention policy is configured as desired.' Below it is a text input field with placeholder text 'Type 'yes' for confirmation'. At the bottom of the dialog are 'OK' and 'Cancel' buttons. The background shows the same access policy interface as the previous screenshot.

7. To enable legal holds, select **Add Policy**. Select **Legal hold** from the drop-down menu.

The screenshot shows the 'wormtestcontainer1 - Access policy' page. A modal dialog is open under the 'Immutable blob storage' section. In the 'Policy type' dropdown, 'Time-based retention' is selected, and 'Legal hold' is shown as the next option, also highlighted with a red border. At the bottom of the dialog are 'OK' and 'Cancel' buttons. The background shows the access policy interface with a table of policies.

8. Create a legal hold with one or more tags.



9. To clear a legal hold, remove the applied legal hold identifier tag.

## Enabling allow protected append blobs writes

- [Portal](#)
- [Azure CLI](#)
- [PowerShell](#)

A screenshot of the Azure portal showing the 'Access policy' settings for a container. The left sidebar shows 'Overview', 'Access Control (IAM)', 'Settings' (which is selected and highlighted in blue), 'Access policy', 'Properties', and 'Metadata'. The main pane is titled 'Immutable blob storage' and contains a 'Policy type' dropdown set to 'Time-based retention', a 'Set retention period for' input field set to '1' with a tooltip explaining it, and a checked checkbox for 'Allow additional appends'. Buttons for 'OK' and 'Cancel' are at the bottom.

## Next steps

[Store business-critical blob data with immutable storage](#)

# Change how a storage account is replicated

3/26/2020 • 9 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) (preview)

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. However, 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 perform a manual migration.

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</sup>	Perform a manual migration  Request a live migration	Perform a manual migration  OR  Switch to GRS/RA-GRS first and then request a live migration <sup>1</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	Perform a manual migration  Request a live migration
...from ZRS	Perform a manual migration	Perform a manual migration	N/A	Use Azure portal, PowerShell, or CLI to change the replication setting <sup>1</sup>

SWITCHING	... TO LRS	...TO GRS/RA-GRS	...TO ZRS	...TO GZRS/RA-GZRS
...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.

#### Caution

If you performed an [account failover](#) for your (RA-)GRS or (RA-)GZRS account, it is configured to be locally redundant in the new primary region. Live migration to ZRS or GZRS for such LRS accounts is not supported. You will need to perform [manual migration](#).

## 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. Select the **Configuration** setting.
3. Update the **Replication** setting.

The screenshot shows the Azure Storage Configuration page for a storage account named "storagesamples". The left sidebar lists various configuration options like Overview, Activity log, and Settings. The main pane shows the current account kind as StorageV2 (general purpose v2), performance as Standard, and secure transfer required as Enabled. The access tier is set to Hot. The Replication section is highlighted with a red box, showing the current setting as "Read-access geo-redundant storage (RA-GRS)". Other options listed are Locally-redundant storage (LRS), Geo-redundant storage (GRS), and Read-access geo-redundant storage (RA-GRS). A note at the bottom states that the current combination of storage account kind, performance, replication and location does not support large file shares.

## Perform a manual migration to ZRS

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.

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

If you need to migrate your storage account from LRS or GRS to ZRS in the primary region with no application downtime, you can request a live migration from Microsoft. During a live migration, you can access data in your storage account, and 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.

ZRS supports general-purpose v2 accounts only, so make sure to upgrade your storage account before you submit 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.

Live migration is supported only for storage accounts that use LRS or GRS replication. If your account uses RA-GRS, then you need to first change your account's replication type to either LRS or GRS before proceeding. This intermediary step removes the secondary read-only endpoint provided by RA-GRS before migration.

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](#). From the portal, select the storage account you want to convert to ZRS.

### 1. Select New Support Request

2. Complete the **Basics** based on your account information. In the **Service** section, select **Storage Account**

Management and the resource you want to convert to ZRS.

3. Select **Next**.
4. Specify the following values in the **Problem** section:
  - **Severity:** Leave the default value as-is.
  - **Problem Type:** Select **Data Migration**.
  - **Category:** Select **Migrate to ZRS**.
  - **Title:** Type a descriptive title, for example, **ZRS account migration**.
  - **Details:** Type additional details in the **Details** box, for example, I would like to migrate to ZRS from [LRS, GRS] in the \_\_ region.
5. Select **Next**.
6. Verify that the contact information is correct on the **Contact information** blade.
7. Select **Create**.

A support person will contact you and provide any assistance you need.

#### NOTE

Live migration is not currently supported for premium file shares. Only manually copying or moving data is currently supported.

GZRS storage accounts do not currently support the archive tier. See [Azure Blob storage: hot, cool, and archive access tiers](#) 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 screenshot shows the 'Configuration' tab of an Azure Storage account's settings. At the top, it says 'The cost of your storage account depends on the usage and the options you choose below.' Below this is a 'Learn more' link. The 'Account kind' is set to 'Storage (general purpose v1)'. A warning message states: '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.' An 'Upgrade' button is present. Under 'Performance', 'Standard' is selected over 'Premium'. A note indicates a 'Secure transfer required' with options 'Disabled' (selected) and 'Enabled'. In the 'Replication' section, 'Zone-redundant storage (ZRS classic)' is chosen, with a note stating: 'The replication setting for a storage accounts using ZRS can't be changed.' The 'Data Lake Storage Gen2' section shows 'Hierarchical namespace' with 'Disabled' selected over 'Enabled'. The entire configuration area is enclosed in a light gray border.

## 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 because your data (in your primary region) is being replicated to your remote secondary region. This charge is a one-time cost at initial setup. After the data is copied, there are no further migration charges. 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](#)
- [Designing highly available applications using read-access geo-redundant storage](#)

# Check the Last Sync Time property for a storage account

2/12/2020 • 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) (preview) 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 an Azure Storage preview module that supports getting geo-replication stats. For example:

```
Install-Module Az.Storage -Repository PSGallery -RequiredVersion 1.1.1-preview -AllowPrerelease -AllowClobber -Force
```

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](#)
- [Designing highly available applications using read-access geo-redundant storage](#)

# Initiate a storage account failover (preview)

3/18/2020 • 3 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 (preview). 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 account failover \(preview\) in Azure Storage](#).

## 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 has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

## Prerequisites

Before you can perform an account failover on your storage account, make sure that you have performed the following step:

- Make sure that your storage account is configured to use either geo-redundant storage (GRS) or read-access geo-redundant storage (RA-GRS). For more information about geo-redundant storage, see [Azure Storage redundancy](#).

## 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 using the `Get-AzStorageAccount` PowerShell cmdlet, and include the `-IncludeGeoReplicationStats` parameter. Then check the `GeoReplicationStats` property for your account. \

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. 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 is dependent on the amount of data being replicated.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

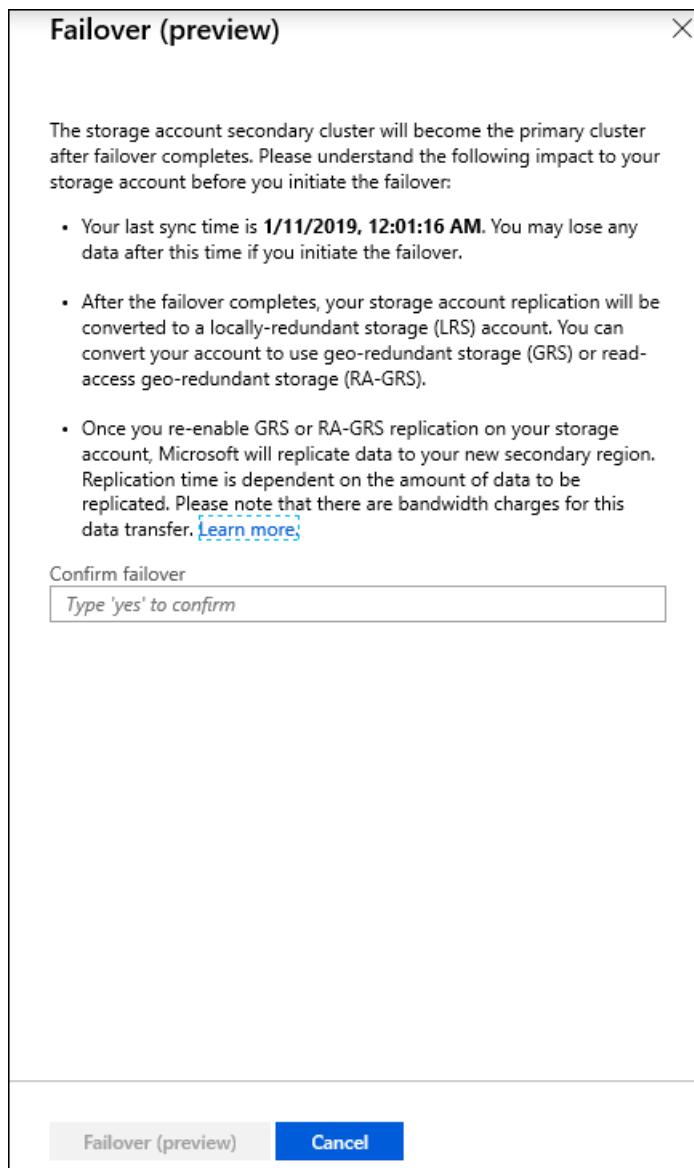
To initiate an account failover from the Azure portal, follow these steps:

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.

LOCATION	DATA CENTER TYPE	STATUS	FAILOVER
US West 2	Primary	Available	-
US West Central	Secondary	Available	-

[Prepare for failover \(preview\)](#)

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.
5. Select **Prepare for failover (preview)**.
6. Review the confirmation dialog. When you are ready, enter **Yes** to confirm and initiate the failover.



## Next steps

- [Disaster recovery and account failover \(preview\) in Azure Storage](#)
- [Designing highly available applications using RA-GRS](#)
- [Tutorial: Build a highly available application with Blob storage](#)

# Soft delete for Azure Storage blobs

3/25/2020 • 17 minutes to read • [Edit Online](#)

Azure Storage now offers soft delete for blob objects so that you can more easily recover your data when it is erroneously modified or deleted by an application or other storage account user.

## NOTE

This feature 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](#).

## How soft delete works

When enabled, soft delete enables you to save and recover your data when blobs or blob snapshots are deleted. This protection extends to blob data that is erased as the result of an overwrite.

When data is deleted, it transitions to a soft deleted state instead of being permanently erased. When soft delete is on and you overwrite data, a soft deleted snapshot is generated to save the state of the overwritten data. Soft deleted objects are invisible unless explicitly listed. You can configure the amount of time soft deleted data is recoverable before it is permanently expired.

Soft delete is backwards compatible, so you don't have to make any changes to your applications to take advantage of the protections this feature affords. However, [data recovery](#) introduces a new **Undelete Blob API**.

## Configuration settings

When you create a new account, soft delete is off by default. Soft delete is also off by default for existing storage accounts. You can toggle the feature on and off at any time during the life of a storage account.

You will still be able to access and recover soft deleted data when the feature is turned off, assuming that soft deleted data was saved when the feature was previously turned on. When you turn on soft delete, you also need to configure the retention period.

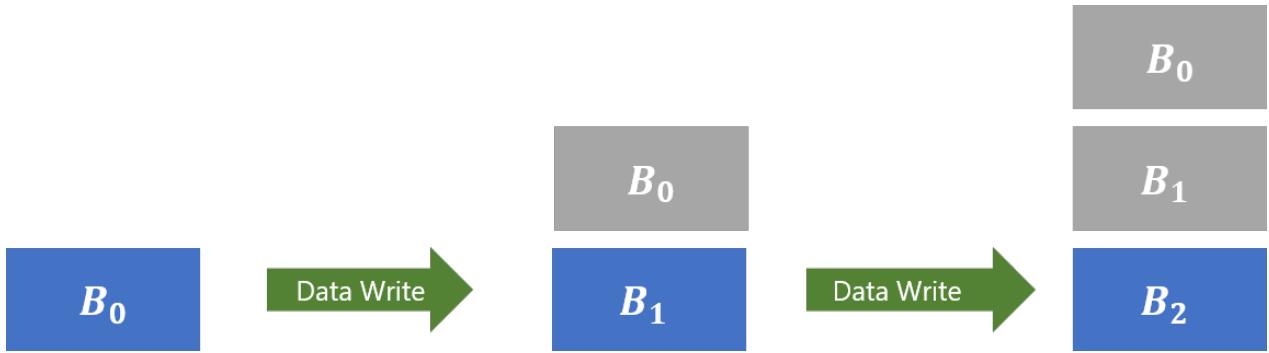
The retention period indicates the amount of time that soft deleted data is stored and available for recovery. For blobs and blob snapshots that are explicitly deleted, the retention period clock starts when the data is deleted. For soft deleted snapshots generated by the soft delete feature when data is overwritten, the clock starts when the snapshot is generated. Currently you can retain soft deleted data for between 1 and 365 days.

You can change the soft delete retention period at any time. An updated retention period will only apply to newly deleted data. Previously deleted data will expire based on the retention period that was configured when that data was deleted. Attempting to delete a soft deleted object will not affect its expiry time.

## Saving deleted data

Soft delete preserves your data in many cases where blobs or blob snapshots are deleted or overwritten.

When a blob is overwritten using **Put Blob**, **Put Block**, **Put Block List**, or **Copy Blob** a snapshot of the blob's state prior to the write operation is automatically generated. This snapshot is a soft deleted snapshot; it is invisible unless soft deleted objects are explicitly listed. See the [Recovery](#) section to learn how to list soft deleted objects.



*Soft deleted data is grey, while active data is blue. More recently written data appears beneath older data. When  $B_0$  is overwritten with  $B_1$ , a soft deleted snapshot of  $B_0$  is generated. When  $B_1$  is overwritten with  $B_2$ , a soft deleted snapshot of  $B_1$  is generated.*

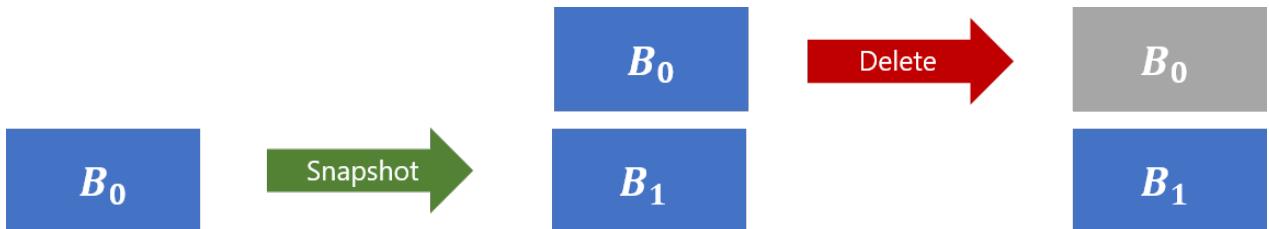
#### NOTE

Soft delete only affords overwrite protection for copy operations when it is turned on for the destination blob's account.

#### NOTE

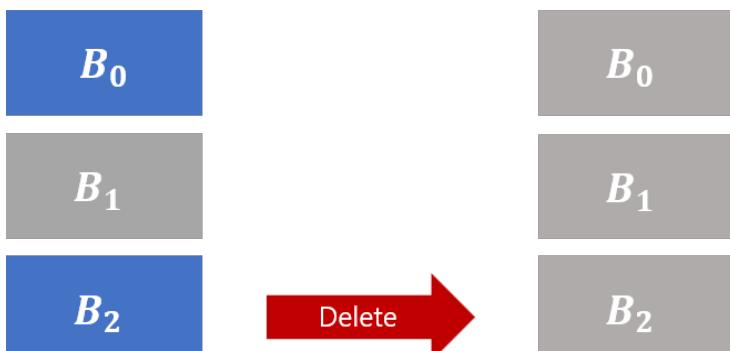
Soft delete does not afford overwrite protection for blobs in the archive tier. If a blob in archive is overwritten with a new blob in any tier, the overwritten blob is permanently expired.

When **Delete Blob** is called on a snapshot, that snapshot is marked as soft deleted. A new snapshot is not generated.



*Soft deleted data is grey, while active data is blue. More recently written data appears beneath older data. When **Snapshot Blob** is called,  $B_0$  becomes a snapshot and  $B_1$  is the active state of the blob. When the  $B_0$  snapshot is deleted, it is marked as soft deleted.*

When **Delete Blob** is called on a base blob (any blob that is not itself a snapshot), that blob is marked as soft deleted. Consistent with previous behavior, calling **Delete Blob** on a blob that has active snapshots returns an error. Calling **Delete Blob** on a blob with soft deleted snapshots does not return an error. You can still delete a blob and all its snapshots in single operation when soft delete is turned on. Doing so marks the base blob and snapshots as soft deleted.



*Soft deleted data is grey, while active data is blue. More recently written data appears beneath older data. Here, a*

**Delete Blob** call is made to delete B2 and all associated snapshots. The active blob, B2, and all associated snapshots are marked as soft deleted.

**NOTE**

When a soft deleted blob is overwritten, a soft deleted snapshot of the blob's state prior to the write operation is automatically generated. The new blob inherits the tier of the overwritten blob.

Soft delete does not save your data in cases of container or account deletes, nor when blob metadata and blob properties are overwritten. To protect a storage account from erroneous deletion, you can configure a lock using the Azure Resource Manager. Please see the Azure Resource Manager article [Lock Resources to Prevent Unexpected Changes](#) to learn more.

The following table details expected behavior when soft delete is turned on:

REST API OPERATION	RESOURCE TYPE	DESCRIPTION	CHANGE IN BEHAVIOR
Delete	Account	Deletes the storage account, including all containers and blobs that it contains.	No change. Containers and blobs in the deleted account are not recoverable.
Delete Container	Container	Deletes the container, including all blobs that it contains.	No change. Blobs in the deleted container are not recoverable.
Put Blob	Block, Append, and Page Blobs	Creates a new blob or replaces an existing blob within a container	If used to replace an existing blob, a snapshot of the blob's state prior to the call is automatically generated. This also applies to a previously soft deleted blob if and only if it is replaced by a blob of the same type (Block, Append, or Page). If it is replaced by a blob of a different type, all existing soft deleted data will be permanently expired.
Delete Blob	Block, Append, and Page Blobs	Marks a blob or blob snapshot for deletion. The blob or snapshot is later deleted during garbage collection	If used to delete a blob snapshot, that snapshot is marked as soft deleted. If used to delete a blob, that blob is marked as soft deleted.
Copy Blob	Block, Append, and Page Blobs	Copies a source blob to a destination blob in the same storage account or in another storage account.	If used to replace an existing blob, a snapshot of the blob's state prior to the call is automatically generated. This also applies to a previously soft deleted blob if and only if it is replaced by a blob of the same type (Block, Append, or Page). If it is replaced by a blob of a different type, all existing soft deleted data will be permanently expired.

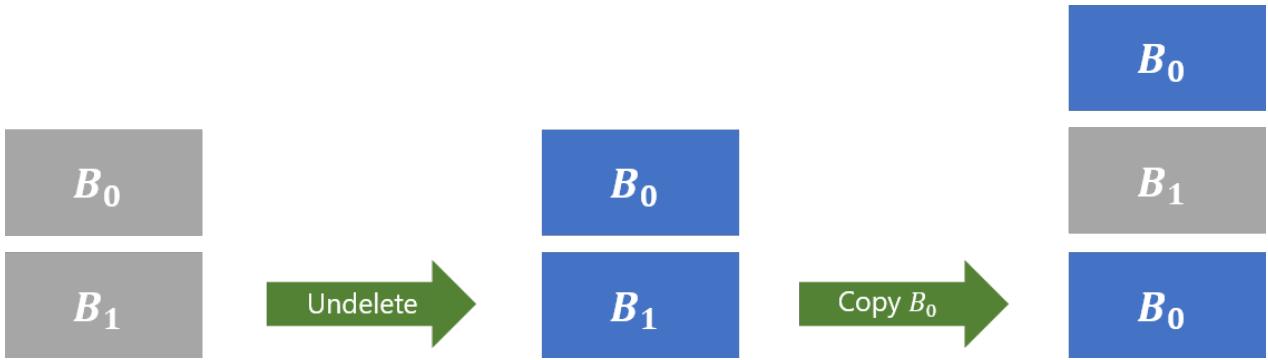
REST API OPERATION	RESOURCE TYPE	DESCRIPTION	CHANGE IN BEHAVIOR
<a href="#">Put Block</a>	Block Blobs	Creates a new block to be committed as part of a block blob.	If used to commit a block to a blob that is active, there is no change. If used to commit a block to a blob that is soft deleted, a new blob is created and a snapshot is automatically generated to capture the state of the soft deleted blob.
<a href="#">Put Block List</a>	Block Blobs	Commits a blob by specifying the set of block IDs that comprise the block blob.	If used to replace an existing blob, a snapshot of the blob's state prior to the call is automatically generated. This also applies to a previously soft deleted blob if and only if it is a Block Blob. If it is replaced by a blob of a different type, all existing soft deleted data will be permanently expired.
<a href="#">Put Page</a>	Page Blobs	Writes a range of pages to a Page Blob.	No change. Page Blob data that is overwritten or cleared using this operation is not saved and is not recoverable.
<a href="#">Append Block</a>	Append Blobs	Writes a block of data to the end of an Append Blob	No change.
<a href="#">Set Blob Properties</a>	Block, Append, and Page Blobs	Sets values for system properties defined for a blob.	No change. Overwritten blob properties are not recoverable.
<a href="#">Set Blob Metadata</a>	Block, Append, and Page Blobs	Sets user-defined metadata for the specified blob as one or more name-value pairs.	No change. Overwritten blob metadata is not recoverable.

It is important to notice that calling "Put Page" to overwrite or clear ranges of a Page Blob will not automatically generate snapshots. Virtual machine disks are backed by Page Blobs and use [Put Page](#) to write data.

## Recovery

Calling the [Undelete Blob](#) operation on a soft deleted base blob restores it and all associated soft deleted snapshots as active. Calling the [Undelete Blob](#) operation on an active base blob restores all associated soft deleted snapshots as active. When snapshots are restored as active, they look like user-generated snapshots; they do not overwrite the base blob.

To restore a blob to a specific soft deleted snapshot, you can call [Undelete Blob](#) on the base blob. Then, you can copy the snapshot over the now-active blob. You can also copy the snapshot to a new blob.



*Soft deleted data is grey, while active data is blue. More recently written data appears beneath older data. Here, **Undelete Blob** is called on blob B, thereby restoring the base blob, B1, and all associated snapshots, here just B0, as active. In the second step, B0 is copied over the base blob. This copy operation generates a soft deleted snapshot of B1.*

To view soft deleted blobs and blob snapshots, you can choose to include deleted data in **List Blobs**. You can choose to view only soft deleted base blobs, or to include soft deleted blob snapshots as well. For all soft deleted data, you can view the time when the data was deleted as well as the number of days before the data will be permanently expired.

### Example

The following is the console output of a .NET script that uploads, overwrites, snapshots, deletes, and restores a blob named *HelloWorld* when soft delete is turned on:

```

Upload:
- HelloWorld (is soft deleted: False, is snapshot: False)

Overwrite:
- HelloWorld (is soft deleted: True, is snapshot: True)
- HelloWorld (is soft deleted: False, is snapshot: False)

Snapshot:
- HelloWorld (is soft deleted: True, is snapshot: True)
- HelloWorld (is soft deleted: False, is snapshot: True)
- HelloWorld (is soft deleted: False, is snapshot: False)

Delete (including snapshots):
- HelloWorld (is soft deleted: True, is snapshot: True)
- HelloWorld (is soft deleted: True, is snapshot: True)
- HelloWorld (is soft deleted: True, is snapshot: False)

Undelete:
- HelloWorld (is soft deleted: False, is snapshot: True)
- HelloWorld (is soft deleted: False, is snapshot: True)
- HelloWorld (is soft deleted: False, is snapshot: False)

Copy a snapshot over the base blob:
- HelloWorld (is soft deleted: False, is snapshot: True)
- HelloWorld (is soft deleted: False, is snapshot: True)
- HelloWorld (is soft deleted: True, is snapshot: True)
- HelloWorld (is soft deleted: False, is snapshot: False)

```

See the [Next steps](#) section for a pointer to the application that produced this output.

## Pricing and billing

All soft deleted data is billed at the same rate as active data. You will not be charged for data that is permanently deleted after the configured retention period. For a deeper dive into snapshots and how they accrue charges, please see [Understanding how snapshots accrue charges](#).

You will not be billed for the transactions related to the automatic generation of snapshots. You will be billed for **Undelete Blob** transactions at the rate for write operations.

For more details on prices for Azure Blob Storage in general, check out the [Azure Blob Storage Pricing Page](#).

When you initially turn on soft delete, we recommend using a small retention period to better understand how the feature will affect your bill.

## Get started

The following steps show how to get started with soft delete.

- [Portal](#)
- [Powershell](#)
- [CLI](#)
- [Python](#)
- [.NET](#)

Enable soft delete for blobs 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 **Blob Service**.
3. Click **Enabled** under **Blob soft delete**
4. Enter the number of days you want to *retain for* under **Retention policies**
5. Choose the **Save** button to confirm your Data Protection settings

The screenshot shows the Azure portal's 'Data protection' settings for a storage account. The left sidebar lists various settings like Access keys, Geo-replication, CORS, Configuration, Encryption, Shared access signature, Firewalls and virtual networks, Private endpoint connection..., Advanced security, Static website, Properties, Locks, and Export template. The 'Data Protection' section is currently selected. The main pane displays the 'Data protection' configuration. It includes a 'Blob soft delete' section where the 'Enabled' button is highlighted. Below it is a 'Retention policies' section with a 'Retain for' field set to 7 days. At the top of the main pane are buttons for Save, Discard, and Refresh.

To view soft deleted blobs, select the **Show deleted blobs** checkbox.

Container Properties							
Location: softdelete							
Search blobs by prefix (case-sensitive)							<input checked="" type="checkbox"/> Show deleted blobs
NAME	STATUS	MODIFIED	ACCESS TIER	BLOB TYPE	SIZE	LEASE STATE	...
Example.html	Active	3/13/2018 1:49:54 PM	Hot (Inferred)	Block blob	79.06 KiB	Available	...
HelloWorld.txt	Active	3/13/2018 1:49:54 PM	Hot (Inferred)	Block blob	13 B	Available	...
Page.vhd	Deleted	3/13/2018 1:49:54 PM	Hot (Inferred)	Page blob	2 KiB		...
Presentation.pptx	Deleted	3/13/2018 1:49:54 PM	Hot (Inferred)	Block blob	369.15 KiB		...

To view soft deleted snapshots for a given blob, select the blob then click **View snapshots**.

Container Properties							
Location: softdelete							
Search blobs by prefix (case-sensitive)							<input checked="" type="checkbox"/> Show deleted blobs
NAME	STATUS	MODIFIED	ACCESS TIER	BLOB TYPE	SIZE	LEASE STATE	...
Example.html	Active	3/13/2018 1:49:54 PM	Hot (Inferred)	Block blob	79.06 KiB	Available	...
HelloWorld.txt	Active	3/13/2018 1:56:12 PM	Hot (Inferred)	Block blob	13 B	Available	...
Page.vhd	Deleted	3/13/2018 1:49:54 PM	Hot (Inferred)	Page blob	2 KiB		...
Presentation.pptx	Deleted	3/13/2018 1:49:54 PM	Hot (Inferred)	Block blob	369.15 KiB		...

Make sure the **Show deleted snapshots** checkbox is selected.

Container Properties						
Schemas						
Filter snapshots						<input checked="" type="checkbox"/> Show deleted snapshots
NAME	STATUS	MODIFIED	BLOB TYPE	CONTENT TYPE	SIZE	...
HelloWorld.txt (3/13/2018 1:55:58 PM)	Active	3/13/2018 1:49:54 PM	Block blob	text/plain	13 B	...
HelloWorld.txt (3/13/2018 1:56:12 PM)	Deleted	3/13/2018 1:49:54 PM	Block blob	text/plain	13 B	...

When you click on a soft deleted blob or snapshot, notice the new blob properties. They indicate when the object was deleted, and how many days are left until the blob or blob snapshot is permanently expired. If the soft deleted object is not a snapshot, you will also have the option to undelete it.

Properties	
NAME	HelloWorld
TYPE	Block blob
SIZE	12 B
ETAG	0x8D5808EFBC566E6
CONTENT-MD5	-
LEASE STATUS	-
LEASE STATE	-
LEASE DURATION	-
STATUS	Deleted
DELETED TIME	3/9/2018 10:54:46 AM
DAYS UNTIL PERMANENT DELETE	6
<a href="#">Undelete</a>	

Remember that undeleting a blob will also undelete all associated snapshots. To undelete soft deleted snapshots for an active blob, click on the blob and select **Undelete all snapshots**.

Overview   Snapshots   Edit blob

**Properties**

URL	<input type="text"/>	
LAST MODIFIED	3/13/2018 1:56:12 PM	
TYPE	Block blob	
SIZE	13 B	
ETAG	0x8D58924DA95E9CF	
CONTENT-MD5	-	
LEASE STATUS	Unlocked	
LEASE STATE	Available	
LEASE DURATION	-	
COPY STATUS	-	
COPY COMPLETION TIME	-	

**Undelete all snapshots**

Once you undelete a blob's snapshots, you can click **Promote** to copy a snapshot over the root blob, thereby restoring the blob to the snapshot.

Overview   Snapshots   Edit blob

Show deleted snapshots

NAME	MODIFIED	BLOB TYPE	CONTENT TYPE	SIZE	...
HelloWorld.txt (3/13/2018 1:55:58 PM)	3/13/2018 1:49:54 PM	Block blob	text/plain	13 B	...
<input checked="" type="checkbox"/> HelloWorld.txt (3/13/2018 1:56:12 PM)	3/13/2018 1:49:54 PM	Block blob	text/plain	13 B	...

**Download**

**Promote**

**Delete snapshot**

## Special considerations

If there is a chance that your data is accidentally modified or deleted by an application or another storage account user, turning on soft delete is recommended. 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.

## FAQ

### For which storage services can I use soft delete?

Currently, soft delete is only available for blob (object) storage.

### Is soft delete available for all storage account types?

Yes, soft delete is available for Blob storage accounts as well as for blobs in general-purpose (both GPv1 and GPv2) storage accounts. Both standard and premium account types are supported. Soft delete is available for unmanaged disks, which are page blobs under the covers. Soft delete is not available for managed disks.

### Is soft delete available for all storage tiers?

Yes, soft delete is available for all storage tiers including hot, cool, and archive. However, soft delete does not afford overwrite protection for blobs in the archive tier.

### Can I use the Set Blob Tier API to tier blobs with soft deleted snapshots?

Yes. The soft deleted snapshots will remain in the original tier, but the base blob will move to the new tier.

**Premium storage accounts have a per blob snapshot limit of 100. Do soft deleted snapshots count toward this limit?**

No, soft deleted snapshots do not count toward this limit.

**Can I turn on soft delete for existing storage accounts?**

Yes, soft delete is configurable for both existing and new storage accounts.

**If I delete an entire account or container with soft delete turned on, will all associated blobs be saved?**

No, if you delete an entire account or container, all associated blobs will be permanently deleted. For more information about protecting a storage account from accidental deletes, see [Lock Resources to Prevent Unexpected Changes](#).

**Can I view capacity metrics for deleted data?**

Soft deleted data is included as a part of your total storage account capacity. For more information on tracking and monitoring storage capacity, see [Storage Analytics](#).

**If I turn off soft delete, will I still be able to access soft deleted data?**

Yes, you will still be able to access and recover unexpired soft deleted data when soft delete is turned off.

**Can I read and copy out soft deleted snapshots of my blob?**

Yes, but you must call Undelete on the blob first.

**Is soft delete available for all blob types?**

Yes, soft delete is available for block blobs, append blobs, and page blobs.

**Is soft delete available for virtual machine disks?**

Soft delete is available for both premium and standard unmanaged disks, which are page blobs under the covers. Soft delete will only help you recover data deleted by **Delete Blob**, **Put Blob**, **Put Block List**, **Put Block** and **Copy Blob** operations. Data overwritten by a call to **Put Page** is not 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 is not a supported scenario.

**Do I need to change my existing applications to use soft delete?**

It is possible to take advantage of soft delete regardless of the API version you are using. However, to list and recover soft deleted blobs and blob snapshots, you will need to use version 2017-07-29 of the [Storage Services REST API](#) or greater. Microsoft recommends always using the latest version of the Azure Storage API.

## Next steps

- [.NET Sample Code](#)
- [Blob Service REST API](#)
- [Azure Storage Replication](#)
- [Designing Highly Available Applications using RA-GRS](#)
- [Disaster recovery and storage account failover \(preview\) in Azure Storage](#)

# Use the Azurite emulator for local Azure Storage development and testing (preview)

1/15/2020 • 9 minutes to read • [Edit Online](#)

The Azurite version 3.2 open-source emulator (preview) provides a free local environment for testing your Azure blob and queue 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 v3 supports APIs implemented by the Azure Blob service.

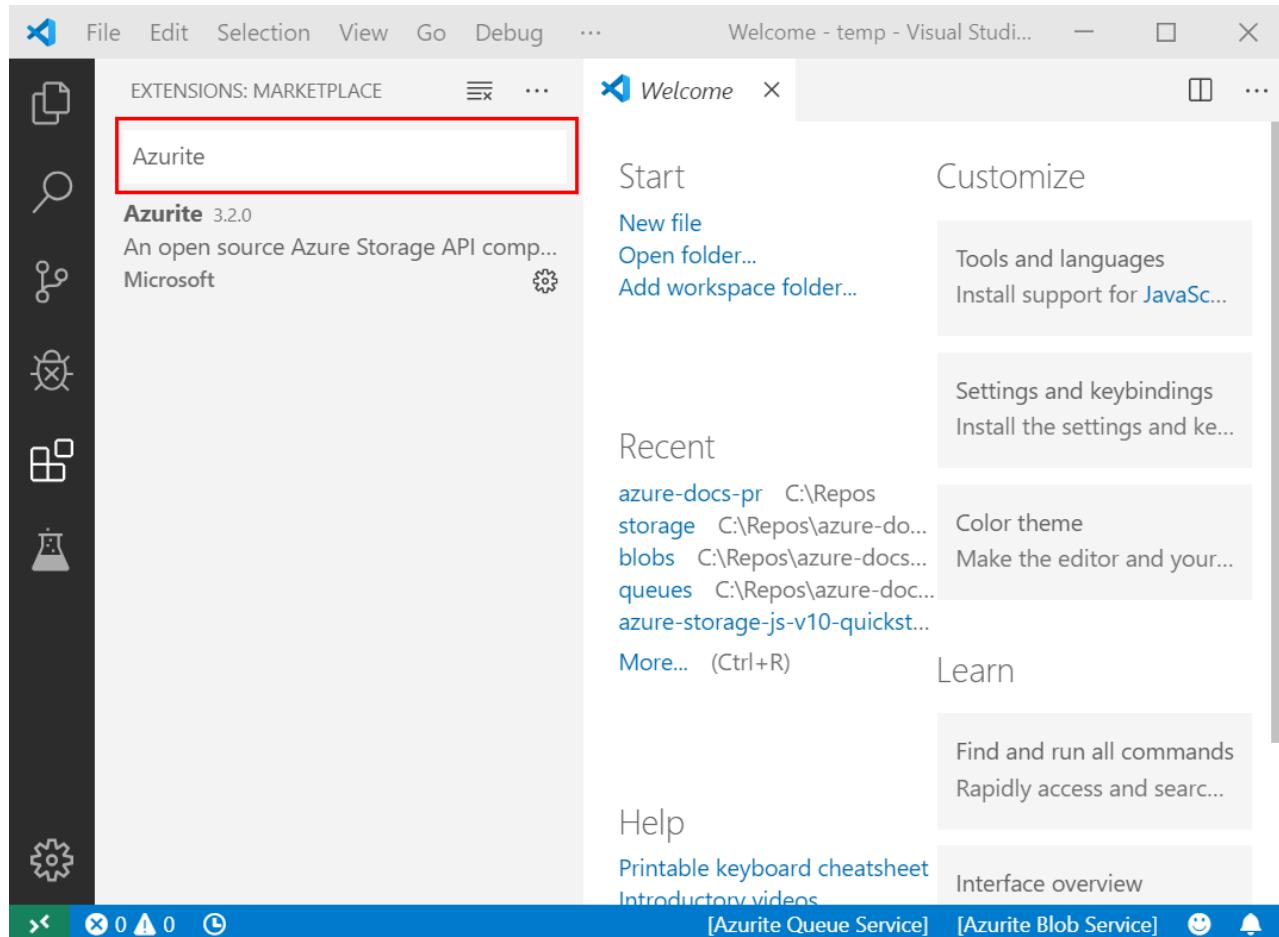
Azurite is the future storage emulator platform. Azurite supersedes the [Azure Storage Emulator](#). Azurite will continue to be updated to support the latest versions of Azure Storage APIs.

There are several different ways to install and run Azurite on your local system:

1. [Install and run the Azurite Visual Studio Code extension](#)
2. [Install and run Azurite by using NPM](#)
3. [Install and run the Azurite Docker image](#)
4. [Clone, build, and run Azurite from the GitHub repository](#)

## Install and run the Azurite Visual Studio Code extension

Within Visual Studio Code, select the EXTENSIONS pane and search for *Azurite* in the EXTENSIONS:MARKETPLACE.



Alternatively, navigate to [VS Code extension market](#) in your browser. Select the **Install** button to open Visual Studio

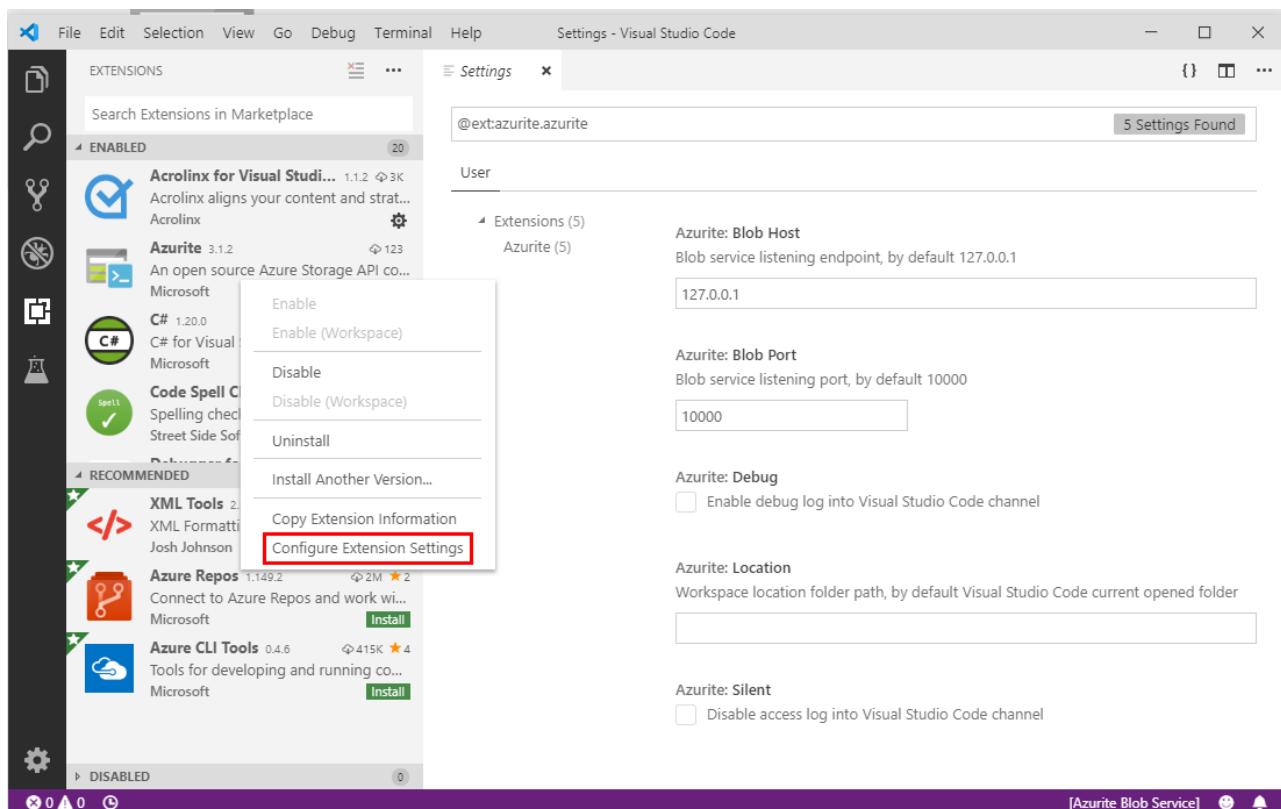
Code and go directly to the Azurite extension page.

You can quickly start or close Azurite by clicking on [Azurite Blob Service] or [Azurite Queue Service] in the VS Code status bar or issuing the following commands in the VS Code command palette. To open the command palette, press F1 in VS Code.

The extension supports the following Visual Studio Code commands:

- **Azurite: Start** - Start all Azurite services
- **Azurite: Close** - Close all Azurite services
- **Azurite: Clean** - Reset all Azurite services persistency data
- **Azurite: Start Blob Service** - Start blob service
- **Azurite: Close Blob Service** - Close blob service
- **Azurite: Clean Blob Service** - Clean blob service
- **Azurite: Start Queue Service** - Start queue service
- **Azurite: Close Queue Service** - Close queue service
- **Azurite: Clean Queue Service** - Clean queue service

To configure Azurite within Visual Studio Code, select the extensions pane. Select the **Manage** (gear) icon for Azurite. Select **Configure Extension Settings**.



The following settings are supported:

- **Azurite: Blob Host** - The Blob service listening endpoint. The default setting is 127.0.0.1.
- **Azurite: Blob Port** - The Blob service listening port. The default port is 10000.
- **Azurite: Debug** - Output the debug log to the Azurite channel. The default value is **false**.
- **Azurite: Location** - The workspace location path. The default is the Visual Studio Code working folder.
- **Azurite: Queue Host** - The Queue service listening endpoint. The default setting is 127.0.0.1.
- **Azurite: Queue Port** - The Queue service listening port. The default port is 10001.
- **Azurite: Silent** - Silent mode disables the access log. The default value is **false**.

## Install and run Azurite by using NPM

This installation method requires that you have [Node.js version 8.0 or later](#) installed. **npm** is the package management tool included with every Node.js installation. After installing Node.js, execute the following **npm** command to install Azurite.

```
npm install -g azurite
```

After installing Azurite, see [Run Azurite from a command-line](#).

## Install and run the Azurite Docker image

Use [DockerHub](#) to pull the [latest Azurite image](#) by using the following command:

```
docker pull mcr.microsoft.com/azure-storage/azurite
```

### Run the Azurite Docker image:

The following command runs the Azurite Docker image. The `-p 10000:10000` parameter redirects requests from host machine's port 10000 to the Docker instance.

```
docker run -p 10000:10000 -p 10001:10001 mcr.microsoft.com/azure-storage/azurite
```

### Specify the workspace location:

In the following example, the `-v c:/azurite:/data` parameter specifies *c:/azurite* as the Azurite persisted data location. The directory, *c:/azurite*, must be created before running the Docker command.

```
docker run -p 10000:10000 -p 10001:10001 -v c:/azurite:/data mcr.microsoft.com/azure-storage/azurite
```

### Run just the blob service

```
docker run -p 10000:10000 mcr.microsoft.com/azure-storage/azurite
azurite-blob --blobHost 0.0.0.0 --blobPort 10000
```

### Set all Azurite parameters:

This example shows how to set all of the command-line parameters. All of the parameters below should be placed on a single command-line.

```
docker run -p 8888:8888
 -p 9999:9999
 -v c:/azurite:/workspace mcr.microsoft.com/azure-storage/azurite azurite
 -l /workspace
 -d /workspace/debug.log
 --blobPort 8888
 --blobHost 0.0.0.0
 --queuePort 9999
 --queueHost 0.0.0.0
```

See [Command-line options](#) for more information about configuring Azurite at start-up.

## Clone, build, and run Azurite from the GitHub repository

This installation method requires that you have [Git](#) installed. Clone the [GitHub repository](#) for the Azurite project by using the following console command.

```
git clone https://github.com/Azure/Azurite.git
```

After cloning the source code, execute following commands from the root of the cloned repo to build and install Azurite.

```
npm install
npm run build
npm install -g
```

After installing and building Azurite, see [Run Azurite from a command-line](#).

## Run Azurite from a command-line

### NOTE

Azurite cannot be run from the command line if you only installed the Visual Studio Code extension. Instead, use the VS Code command palette. For more information, see [Install and run the Azurite Visual Studio Code extension](#).

To get started immediately with the command-line, create a directory called `c:\azurite`, then launch Azurite by issuing the following command:

```
azurite --silent --location c:\azurite --debug c:\azurite\debug.log
```

This command tells Azurite to store all data in a particular directory, `c:\azurite`. If the `--location` option is omitted, it will use the current working directory.

## Command-line options

This section details the command-line switches available when launching Azurite. All command-line switches are optional.

```
C:\Azurite> azurite [--blobHost <IP address>] [--blobPort <port address>]
[-d | --debug <log file path>] [-l | --location <workspace path>]
[--queueHost <IP address>] [--queuePort <port address>]
[-s | --silent] [-h | --help]
```

The `-d` is a shortcut for `--debug`, `-l` switch is a shortcut for `--location`, `-s` is a shortcut for `--silent`, and `-h` is a shortcut for `--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.

## Workspace path

**Optional** Azurite stores data to the local disk during execution. Use the **--location** switch to specify a path as the workspace location. By default, the current process working directory will be used.

```
azurite --location c:\azurite
```

```
azurite -l 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 **--silent** switch.

```
azurite --silent
```

```
azurite -s
```

## 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 **--debug** switch.

```
azurite --debug path/debug.log
```

```
azurite -d 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 **--loose** switch.

```
azurite --loose
```

Note the capital 'L' shortcut switch:

```
azurite -L
```

# 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 Shared Key and shared access signatures (SAS). Azurite also supports anonymous access to public containers.

### Well-known storage account and key

You can use the following account name and key with Azurite. This is the same well-known account and key used by the legacy Azure storage emulator.

- Account name: `devstoreaccount1`
- Account key: `Eby8vdM02xN0cqFlqUwJPLlmEt1CDXJ1OUzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==`

#### NOTE

In addition to SharedKey authentication, Azurite supports account and service SAS authentication. Anonymous access is also available when a container is set to allow public access.

### Connection string

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>
```

For more information, see [Configure Azure Storage connection strings](#).

### 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"
```

Or use multiple storage accounts with 2 keys each:

```
set 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.

#### NOTE

Update the connection string accordingly when using custom account names and keys.

## NOTE

Use the `export` keyword to set environment variables in a Linux environment, use `set` in Windows.

## Storage Explorer

In Azure Storage Explorer, connect to Azurite by clicking the **Add Account** icon, then select **Attach to a local emulator** and click **Connect**.

## 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
```

Since 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 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
```

### Scaling and performance

Azurite is not a scalable storage service and does not support a large number of concurrent 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
```

## 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.

# Use the Azure storage emulator for development and testing

3/30/2020 • 17 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.

## 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. If you need a storage emulator for Linux, one option is the community maintained, open-source storage emulator [Azurite](#).

### 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 reinitializes 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 SSMS 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 storage 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 storage emulator. They are:

```
Account name: devstoreaccount1
Account key: Eby8vdM02xN0cqFlqUwJPLlmEt1CDXJ1OUzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==
```

### NOTE

The authentication key supported by the storage 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 storage emulator. You should not use the development account with production data.

The storage 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 a shortcut

The easiest way to connect to the storage emulator 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 to the storage emulator in an *app.config* file:

```
<appSettings>
 <add key="StorageConnectionString" value="UseDevelopmentStorage=true" />
</appSettings>
```

### Connect to the emulator account using the well-known account name and key

To create a connection string that references the emulator account name and key, you must specify the endpoints for each of the services you wish to use from the emulator in the connection string. This is necessary so that the connection string will reference the emulator endpoints, which are different than those for a production storage account. For example, the value of your connection string will look like this:

```
DefaultEndpointsProtocol=http;AccountName=devstoreaccount1;
AccountKey=Eby8vdM02xN0cqFlqUwJPLlmEt1CDXJ1OUzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==;
BlobEndpoint=http://127.0.0.1:10000/devstoreaccount1;
TableEndpoint=http://127.0.0.1:10002/devstoreaccount1;
QueueEndpoint=http://127.0.0.1:10001/devstoreaccount1;
```

This value is identical to the shortcut shown above, `UseDevelopmentStorage=true`.

### Specify an HTTP proxy

You can also specify an HTTP proxy to use when you're testing your service against the storage emulator. This can be useful for observing HTTP requests and responses while you're debugging operations against the storage services. To specify a proxy, add the `DevelopmentStorageProxyUri` option to the connection string, and set its value to the proxy URI. For example, here is a connection string that points to the storage emulator and configures an HTTP proxy:

```
UseDevelopmentStorage=true;DevelopmentStorageProxyUri=http://myProxyUri
```

For more information on connection strings, see [Configure Azure Storage connection strings](#).

## Authorize with a shared access signature

## NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

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>	<i>-Reprocess</i> : 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]	-server <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 FeatureNotSupportedException (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 `VersionNotSupportedException` (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.

# Host a static website in Azure Storage

3/25/2020 • 6 minutes to read • [Edit Online](#)

You can serve static content (HTML, CSS, JavaScript, and image files) directly from a container in an Azure Storage GPv2 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 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 (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**. The Azure portal now displays your static website endpoint.

The screenshot shows the Azure Storage portal interface for managing a static website. On the left, a sidebar lists various storage-related settings like Access keys, Geo-replication, and CORS. The 'Static website' option is currently selected and highlighted in grey. The main panel displays information about enabling static websites on the blob service, mentioning that webpages may include static content and client-side scripts. It shows that the 'Static website' feature is currently 'Enabled'. Below this, it indicates that an Azure Storage container has been created to host the static website, with the primary endpoint set to `https://contoso.web.core.windows.net/`. The index document name is set to `index.html` and the error document path is set to `404.html`. There are 'Save' and 'Discard' buttons at the top right of the main panel.

## Upload files

- [Portal](#)
- [Azure CLI](#)
- [PowerShell](#)

These instructions show you how to upload files by using the version of Storage Explorer that appears in the Azure portal. However, you can also use the version of [Storage Explorer](#) that runs outside of the Azure portal. You could 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. Select **Storage Explorer (preview)**.
2. Expand the **BLOB CONTAINERS** node, and then select the `$web` container.
3. Choose the **Upload** button to upload files.

The screenshot shows the Azure Storage Explorer (preview) interface. On the left, there's a sidebar with various options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Data transfer, Events, and Storage Explorer (preview). The Storage Explorer (preview) option is highlighted with a red box. The main area shows a tree view of blob containers under 'BLOB CONTAINERS'. The '\$web' container is selected and highlighted with a red box. Inside '\$web', there are five containers named Container1 through Container5. Below '\$web', there are sections for FILE SHARES, QUEUES, and TABLES. On the right, there's a list of files in the '\$web' container. The file '404.html' is selected and highlighted with a red box. The list includes '404.html' (Hot (inferred)), 'index.html' (Hot (inferred)), and 'myMarkdownFile.md' (Hot (inferred)). At the top right, there are buttons for Upload, Download, Open, and a search bar.

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`.

The screenshot shows the Azure Storage Explorer (preview) interface with the '\$web' container selected. In the list of files, '404.html' is selected. The table below shows the file properties. The 'CONTENT TYPE' column is highlighted with a red box. The table has columns: NAME, ACCESS TIER, ACCESS TIER LAST MODIFIED, LAST MODIFIED, BLOB TYPE, and CONTENT TYPE. The data is as follows:

NAME	ACCESS TIER	ACCESS TIER LAST MODIFIED	LAST MODIFIED	BLOB TYPE	CONTENT TYPE
404.html	Hot (inferred)		5/16/2019, 11:54:07 AM	Block Blob	text/html
index.html	Hot (inferred)		5/16/2019, 11:54:07 AM	Block Blob	text/html
myMarkdownFile.md	Hot (inferred)		3/4/2020, 2:58:17 PM	Block Blob	application/octet-stream

#### NOTE

Storage Explorer automatically sets this property 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. To set this property, right-click the file, and then click **Properties**.

## Find the website URL by using the Azure portal

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.

Search resources, services, and docs

Home > Contoso - Static website

Contoso - Static website

Storage account

Save Discard

Configuring the blob service for static website hosting enables you to host static content in your storage account. Webpages may include static content and client-side scripts. Server-side scripting is not supported in Azure Storage. [Learn more](#)

Static website

Disabled Enabled

An Azure Storage container has been created to host your static website.

\$web

Primary endpoint <https://contosoblobaccount.z22.web.core.windows.net/>

Index document name index.html

Error document path 404.html

Overview Activity log Access control (IAM) Tags Diagnose and solve problems Data transfer Events Storage Explorer (preview)

Settings

Access keys Geo-replication

## 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.

Monitoring

Insights (preview)

Alerts

Metrics

Workbooks

Advisor recommendations

### 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**.

Last 24 hours (Automatic)

Time range

Last 30 minutes Last 3 days  
Last 4 hours Last 7 days  
Last 12 hours Last 30 days  
Last 24 hours Custom  
Last 48 hours

Time granularity Automatic

Show time as UTC/GMT Local

Apply Cancel

3. Select **Blob** from the *Namespace* drop down.

The screenshot shows the Azure Metrics blade. At the top, there are four dropdown menus: 'RESOURCE' set to 'contosodemo', 'METRIC NAMESPACE' set to 'Select namespace', 'METRIC' set to 'Select metric', and 'AGGREGATION' set to 'Select value(s)'. A blue box highlights the 'METRIC NAMESPACE' dropdown, which is open to show a list of options: Account, Blob, File, Queue, and Table. The 'Blob' option is selected and highlighted with a blue background.

4. Then select the **Egress** metric.

The screenshot shows the Azure Metrics blade. The 'RESOURCE' dropdown is set to 'contosodemo'. The 'METRIC NAMESPACE' dropdown is set to 'Blob'. The 'METRIC' dropdown is set to 'Select metric'. The 'AGGREGATION' dropdown is set to 'Select value(s)'. A blue box highlights the 'METRIC' dropdown, which is open to show a list of metrics categorized under 'CAPACITY' and 'TRANSACTION'. Under 'TRANSACTION', the 'Egress' metric is selected and highlighted with a blue background.

5. Select **Sum** from the *Aggregation* selector.

The screenshot shows the Azure Metrics blade. The 'RESOURCE' dropdown is set to 'contosodemo'. The 'METRIC NAMESPACE' dropdown is set to 'Blob'. The 'METRIC' dropdown is set to 'Egress'. The 'AGGREGATION' dropdown is set to 'Sum'. A blue box highlights the 'AGGREGATION' dropdown, which is open to show four aggregation functions: Sum, Avg, Min, and Max. The 'Sum' function is selected and highlighted with a blue background.

6. Click the **Add filter** button and choose **API name** from the *Property* selector.

The screenshot shows the Azure Metrics blade with a filter added. The filter is 'contosodemo, Egress, Sum'. A blue box highlights the 'Add metric' button, which is used to add a new filter. The 'PROPERTY' dropdown is set to 'Select property' and the 'VALUES' dropdown is set to 'Select value(s)'. A blue box highlights the 'VALUES' dropdown, which is open to show three options: Geo type, API name, and Authentication. The 'API name' option is selected and highlighted with a blue background.

7. Check the box next to **GetWebContent** in the *Values* selector to populate the metrics report.

The screenshot shows the Azure Metrics Explorer interface. At the top, there is a search bar with the text "contosodemo, Egress, Sum". To the right of the search bar are two buttons: "Add metric" and a refresh icon. Below the search bar, there is a filter section with a "PROPERTY" dropdown set to "API name" and a "VALUES" dropdown. The "VALUES" dropdown is expanded, showing a list of API members: GetBlobServiceProperties, BlobPreflightRequest, SetBlobServiceProperties, GetContainerProperties, DeleteBlob, **GetWebContent** (which is checked), GetBlobProperties, and GetContainerACL. The "GetWebContent" option is highlighted with a blue selection bar.

#### 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

4/7/2020 • 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 **Blob Service** menu, select **Azure CDN** to open the Azure CDN page:

The screenshot shows the 'New endpoint' configuration page. It includes fields for creating a new CDN profile (radio button selected), choosing a pricing tier (dropdown menu), naming the CDN endpoint (text input field ending in '.azureedge.net'), and specifying the origin hostname (text input field). A 'Create' button is at the bottom.

New endpoint

CDN profile ⓘ  
 Create new  Use existing

Pricing tier ([View full pricing details](#)) \*

CDN endpoint name \*

Origin hostname ⓘ  
storagesamplestatic.z5.web.core.windows.net

Create

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 \*

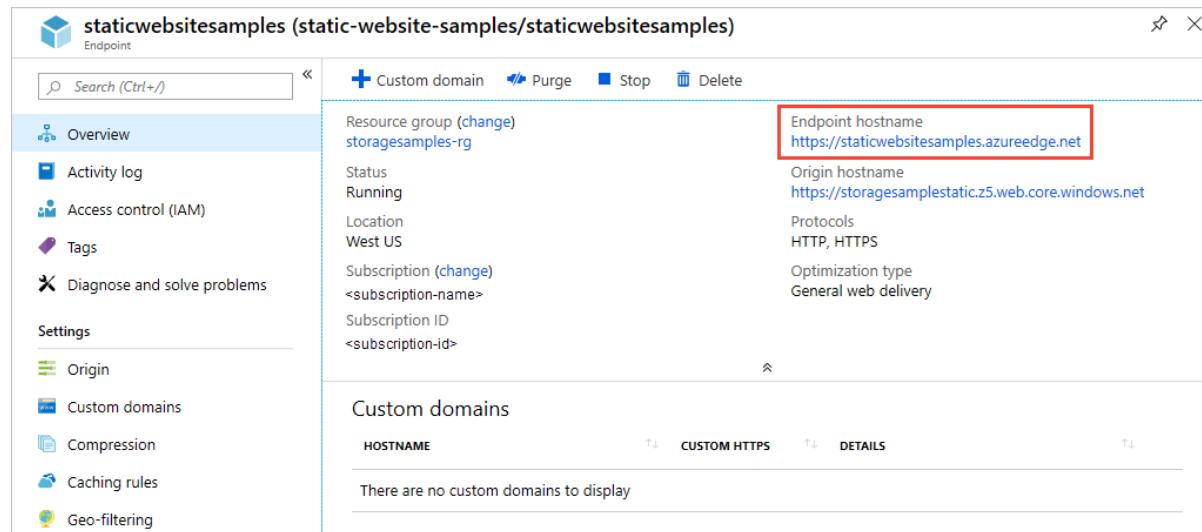
staticwebsitesamples.azureedge.net

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

Overview

Resource group (change)  
storagesamples-rg

Status  
Running

Location  
West US

Subscription (change)  
<subscription-name>

Subscription ID  
<subscription-id>

Custom domains

HOSTNAME CUSTOM HTTPS DETAILS

Endpoint hostname  
`https://staticwebsitesamples.azureedge.net`

Origin hostname  
`https://storagesamplestatic.z5.web.core.windows.net`

Protocols  
HTTP, HTTPS

Optimization type  
General web delivery

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 like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, and Settings. Under Settings, the 'Origin' option is selected and highlighted in blue. The main content area is titled 'Origin type' and shows 'Custom origin' selected. Below it, 'Origin hostname' is set to 'storagesamplestatic.z5.web.core.windows.net'. Other settings include 'Origin host header' (set to 'storagesamplestatic.z5.web.core.windows.net'), 'Origin path' (set to '/Path'), 'Protocol' (HTTP checked, HTTPS checked), 'Origin port' (80 for HTTP, 443 for HTTPS), and 'Origin port' (80 for HTTP, 443 for HTTPS). The 'Save' and 'Discard' buttons 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](#).

# Map a custom domain to an Azure Blob Storage endpoint

3/13/2020 • 9 minutes to read • [Edit Online](#)

You can map a custom domain to a blob service endpoint or a [static website endpoint](#).

## NOTE

This feature 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](#).

## 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 **Properties**.
3. Copy the value of the **Primary Blob Service Endpoint** or the **Primary static website endpoint** to a

text file.

4. Remove the protocol identifier (*e.g.*, 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.windows.net</code>	<code>mystorageaccount.blob.core.windows.net</code>
static website	<code>https://mystorageaccount.z5.web.core.windows.net</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 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.

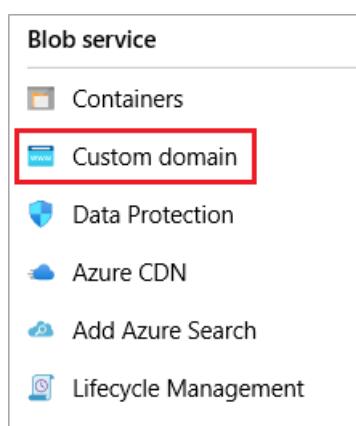
You might have to go to an advanced settings page and look for **CNAME, Alias, or Subdomains**.

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

1. In the [Azure portal](#), go to your storage account.
2. In the menu pane, under **Blob Service**, select **Custom domain**.



The **Custom domain** pane opens.

3. 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`.

4. 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 following 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 a 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 **Properties**.
3. Copy the value of the **Primary Blob Service Endpoint** or the **Primary static website endpoint** to a text file.
4. Remove the protocol identifier (*e.g.*, 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.windows.net</code>	<code>mystorageaccount.blob.core.windows.net</code>
static website	<code>https://mystorageaccount.z5.web.core.windows.net</code>	<code>mystorageaccount.z5.web.core.windows.net</code>

Set this value aside for later.

#### Step 2: Create a 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`.

4. 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.

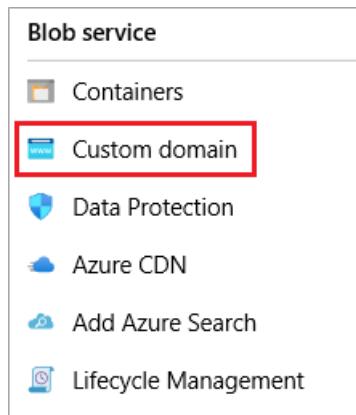
Your custom domain has been verified by Azure, but traffic to your domain is not yet being routed to your storage account.

**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.

1. In the [Azure portal](#), go to your storage account.

2. In the menu pane, under **Blob Service**, select **Custom domain**.



The **Custom domain** pane opens.

3. 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`.

4. Select the **Use indirect CNAME validation** check box.

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: 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 URL: `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](#)
- [Azure CLI](#)
- [PowerShell](#)

To remove the custom domain setting, do the following:

1. In the [Azure portal](#), go to your storage account.
2. In the menu pane, under **Blob Service**, select **Custom domain**.  
The **Custom domain** pane opens.
3. Clear the contents of the text box that contains your custom domain name.
4. 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.

To map a custom domain and enable HTTPS access, do the following:

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](#).

## Next steps

- [Learn about static website hosting in Azure Blob storage](#)

# Quickstart: Route storage events to web endpoint with Azure CLI

4/6/2020 • 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.

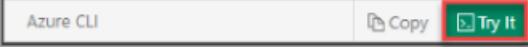
When you complete the steps described in this article, you see that the event data has been sent to the web app.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## 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 CLI locally, this article requires that you're running the latest version of Azure CLI (2.0.70 or later). To find the version, run `az --version`. If you need to install or upgrade, see [Install Azure CLI](#).

If you aren't using Cloud Shell, you must first sign in using `az login`.

## 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 group deployment 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 Event Grid resource provider

If you haven't previously used Event Grid in your Azure subscription, you may need to register the Event Grid resource provider. Run the following command to register the provider:

```
az provider register --namespace Microsoft.EventGrid
```

It may 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.

The screenshot shows the Azure Event Grid Viewer interface. At the top, there's a logo and the title "Azure Event Grid Viewer". Below the title, there's a section titled "Event Type". Under "Event Type", there's a list item for "Microsoft.EventGrid.SubscriptionValidationEvent". To the left of this list item is a small blue icon with a white eye symbol. Below the list item, the event data is displayed as JSON. The JSON data is as follows:

```

[{
 "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_ACCESS_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-
xxxxxxxx/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/testblob1.txt",
 "sequencer": "000000000000EB0000000000046199",
 "storageDiagnostics": {
 "batchId": "dfea416-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](#)

# Process change feed in Azure Blob Storage (Preview)

11/14/2019 • 6 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 \(Preview\)](#).

## NOTE

The change feed is in public preview, and is available in the **westcentralus** and **westus2** regions. To learn more about this feature along with known issues and limitations, see [Change feed support in Azure Blob Storage](#). The change feed processor library is subject to change between now and when this library becomes generally available.

## Get the blob change feed processor library

1. In Visual Studio, add the URL

<https://azuresdkartifacts.blob.core.windows.net/azuresdkpartnerdrops/index.json> to your NuGet package sources.

To learn how, see [package sources](#).

2. In the NuGet Package Manager, Find the **Microsoft.Azure.Storage.Changefeed** package, and install it to your project.

To learn how, see [Find and install a package](#).

## Connect to the storage account

Parse the connection string by calling the [CloudStorageAccount.TryParse](#) method.

Then, create an object that represents Blob Storage in your storage account by calling the [CloudStorageAccount.CreateCloudBlobClient](#) method.

```
public bool GetBlobClient(ref CloudBlobClient cloudBlobClient, string storageConnectionString)
{
 if (CloudStorageAccount.TryParse
 (storageConnectionString, out CloudStorageAccount storageAccount))
 {
 cloudBlobClient = storageAccount.CreateCloudBlobClient();

 return true;
 }
 else
 {
 return false;
 }
}
```

## Initialize the change feed

Add the following using statements to the top of your code file.

```
using Avro.Generic;
using ChangeFeedClient;
```

Then, create an instance of the **ChangeFeed** class by calling the **GetContainerReference** method. Pass in the name of the change feed container.

```
public async Task<ChangeFeed> GetChangeFeed(CloudBlobClient cloudBlobClient)
{
 CloudBlobContainer changeFeedContainer =
 cloudBlobClient.GetContainerReference("$blobchangefeed");

 ChangeFeed changeFeed = new ChangeFeed(changeFeedContainer);
 await changeFeed.InitializeAsync();

 return changeFeed;
}
```

## Reading 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.

The simplest way to read records is to create an instance of the **ChangeFeedReader** class.

This example iterates through all records in the change feed, and then prints to the console a few values from each record.

```
public async Task ProcessRecords(ChangeFeed changeFeed)
{
 ChangeFeedReader processor = await changeFeed.CreateChangeFeedReaderAsync();

 ChangeFeedRecord currentRecord = null;
 do
 {
 currentRecord = await processor.GetNextItemAsync();

 if (currentRecord != null)
 {
 string subject = currentRecord.record["subject"].ToString();
 string eventType = ((GenericEnum)currentRecord.record["eventType"]).Value;
 string api = ((GenericEnum)((GenericRecord)currentRecord.record["data"])["api"]).Value;

 Console.WriteLine("Subject: " + subject + "\n" +
 "Event Type: " + eventType + "\n" +
 "Api: " + api);
 }
 } while (currentRecord != null);
}
```

## Resuming reading records from a saved position

You can choose to save your read position in your change feed and resume iterating the records at a future time. You can save the state of your iteration of the change feed at any time using the **ChangeFeedReader.SerializeState()** method. The state is a **string** and your application can save that state based on your application's design (For example: to a database or a file).

```
string currentReadState = processor.SerializeState();
```

You can continue iterating through records from the last state by creating the **ChangeFeedReader** using the **CreateChangeFeedReaderFromPointerAsync** method.

```
public async Task ProcessRecordsFromLastPosition(ChangeFeed changeFeed, string lastReadState)
{
 ChangeFeedReader processor = await changeFeed.CreateChangeFeedReaderFromPointerAsync(lastReadState);

 ChangeFeedRecord currentRecord = null;
 do
 {
 currentRecord = await processor.GetNextItemAsync();

 if (currentRecord != null)
 {
 string subject = currentRecord.record["subject"].ToString();
 string eventType = ((GenericEnum)currentRecord.record["eventType"]).Value;
 string api = ((GenericEnum)((GenericRecord)currentRecord.record["data"])["api"]).Value;

 Console.WriteLine("Subject: " + subject + "\n" +
 "Event Type: " + eventType + "\n" +
 "Api: " + api);
 }
 } while (currentRecord != null);
}
```

## Stream processing of records

You can choose to process change feed records as they arrive. See [Specifications](#).

```

public async Task ProcessRecordsStream(ChangeFeed changeFeed, int waitTimeMs)
{
 ChangeFeedReader processor = await changeFeed.CreateChangeFeedReaderAsync();

 ChangeFeedRecord currentRecord = null;
 while (true)
 {
 do
 {
 currentRecord = await processor.GetNextItemAsync();

 if (currentRecord != null)
 {
 string subject = currentRecord.record["subject"].ToString();
 string eventType = ((GenericEnum)currentRecord.record["eventType"]).Value;
 string api = ((GenericEnum)((GenericRecord)currentRecord.record["data"])["api"]).Value;

 Console.WriteLine("Subject: " + subject + "\n" +
 "Event Type: " + eventType + "\n" +
 "Api: " + api);
 }
 } while (currentRecord != null);

 await Task.Delay(waitTimeMs);
 }
}

```

## Reading records within a time range

The change feed is organized into hourly segments based on the change event time. See [Specifications](#). You can read records from change feed segments that fall within a specific time range.

This example gets the starting times of all segments. Then, it iterates through that list until the starting time is either beyond the time of the last consumable segment or beyond the ending time of the desired range.

### Selecting segments for a time range

```

public async Task<List<DateTimeOffset>> GetChangeFeedSegmentRefsForTimeRange
 (ChangeFeed changeFeed, DateTimeOffset startTime, DateTimeOffset endTime)
{
 List<DateTimeOffset> result = new List<DateTimeOffset>();

 DateTimeOffset stAdj = startTime.AddMinutes(-15);
 DateTimeOffset enAdj = endTime.AddMinutes(15);

 DateTimeOffset lastConsumable = (DateTimeOffset)changeFeed.LastConsumable;

 List<DateTimeOffset> segments =
 (await changeFeed.ListAvailableSegmentTimesAsync()).ToList();

 foreach (var segmentStart in segments)
 {
 if (lastConsumable.CompareTo(segmentStart) < 0)
 {
 break;
 }

 if (enAdj.CompareTo(segmentStart) < 0)
 {
 break;
 }

 DateTimeOffset segmentEnd = segmentStart.AddMinutes(60);

 bool overlaps = stAdj.CompareTo(segmentEnd) < 0 &&
 segmentStart.CompareTo(enAdj) < 0;

 if (overlaps)
 {
 result.Add(segmentStart);
 }
 }

 return result;
}

```

## Reading records in a segment

You can read records from individual segments or ranges of segments.

```

public async Task ProcessRecordsInSegment(ChangeFeed changeFeed, DateTimeOffset segmentOffset)
{
 ChangeFeedSegment segment = new ChangeFeedSegment(segmentOffset, changeFeed);
 await segment.InitializeAsync();

 ChangeFeedSegmentReader processor = await segment.CreateChangeFeedSegmentReaderAsync();

 ChangeFeedRecord currentRecord = null;
 do
 {
 currentRecord = await processor.GetNextItemAsync();

 if (currentRecord != null)
 {
 string subject = currentRecord.record["subject"].ToString();
 string eventType = ((GenericEnum)currentRecord.record["eventType"]).Value;
 string api = ((GenericEnum)((GenericRecord)currentRecord.record["data"])["api"]).Value;

 Console.WriteLine("Subject: " + subject + "\n" +
 "Event Type: " + eventType + "\n" +
 "Api: " + api);
 }
 } while (currentRecord != null);
}

```

## Read records starting from a time

You can read the records of the change feed from a starting segment till the end. Similar to reading records within a time range, you can list the segments and choose a segment to start iterating from.

This example gets the [DateTimeOffset](#) of the first segment to process.

```

public async Task<DateTimeOffset> GetChangeFeedSegmentRefAfterTime
 (ChangeFeed changeFeed, DateTimeOffset timestamp)
{
 DateTimeOffset result = new DateTimeOffset();

 DateTimeOffset lastConsumable = (DateTimeOffset)changeFeed.LastConsumable;
 DateTimeOffset lastConsumableEnd = lastConsumable.AddMinutes(60);

 DateTimeOffset timestampAdj = timestamp.AddMinutes(-15);

 if (lastConsumableEnd.CompareTo(timestampAdj) < 0)
 {
 return result;
 }

 List<DateTimeOffset> segments = (await changeFeed.ListAvailableSegmentTimesAsync()).ToList();
 foreach (var segmentStart in segments)
 {
 DateTimeOffset segmentEnd = segmentStart.AddMinutes(60);
 if (timestampAdj.CompareTo(segmentEnd) <= 0)
 {
 result = segmentStart;
 break;
 }
 }

 return result;
}

```

This example processes change feed records starting from the [DateTimeOffset](#) of a starting segment.

```

public async Task ProcessRecordsStartingFromSegment(ChangeFeed changeFeed, DateTimeOffset segmentStart)
{
 TimeSpan waitTime = new TimeSpan(60 * 1000);

 ChangeFeedSegment segment = new ChangeFeedSegment(segmentStart, changeFeed);

 await segment.InitializeAsync();

 while (true)
 {
 while (!await IsSegmentConsumableAsync(changeFeed, segment))
 {
 await Task.Delay(waitTime);
 }

 ChangeFeedSegmentReader reader = await segment.CreateChangeFeedSegmentReaderAsync();

 do
 {
 await reader.CheckForFinalizationAsync();

 ChangeFeedRecord currentItem = null;
 do
 {
 currentItem = await reader.GetNextItemAsync();
 if (currentItem != null)
 {
 string subject = currentItem.record["subject"].ToString();
 string eventType = ((GenericEnum)currentItem.record["eventType"]).Value;
 string api = ((GenericEnum)((GenericRecord)currentItem.record["data"])["api"]).Value;

 Console.WriteLine("Subject: " + subject + "\n" +
 "Event Type: " + eventType + "\n" +
 "Api: " + api);
 }
 } while (currentItem != null);

 if (segment.timeWindowStatus != ChangefeedSegmentStatus.Finalized)
 {
 await Task.Delay(waitTime);
 }
 } while (segment.timeWindowStatus != ChangefeedSegmentStatus.Finalized);

 segment = await segment.GetNextSegmentAsync(); // TODO: What if next window doesn't yet exist?
 await segment.InitializeAsync(); // Should update status, shard list.
 }
}

private async Task<bool> IsSegmentConsumableAsync(ChangeFeed changeFeed, ChangeFeedSegment segment)
{
 if (changeFeed.LastConsumable >= segment.startTime)
 {
 return true;
 }
 await changeFeed.InitializeAsync();
 return changeFeed.LastConsumable >= segment.startTime;
}

```

**TIP**

A segment of the can have change feed logs in one or more *chunkFilePath*. In case of multiple *chunkFilePath* the system has internally partitioned the records into multiple shards to manage publishing throughput. It is guaranteed that each partition of the segment will contain changes for mutually exclusive blobs and can be processed independently without violating the ordering. You can use the **ChangeFeedSegmentShardReader** class to iterate through records at the shard level if that's most efficient for your scenario.

## Next steps

Learn more about change feed logs. See [Change feed in Azure Blob Storage \(Preview\)](#)

# Get started with AzCopy

4/6/2020 • 11 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 files.

## 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 \(tar\)](#)
- [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.

## NOTE

If you want to copy data to and from your [Azure Table storage](#) service, then install [AzCopy version 7.3](#).

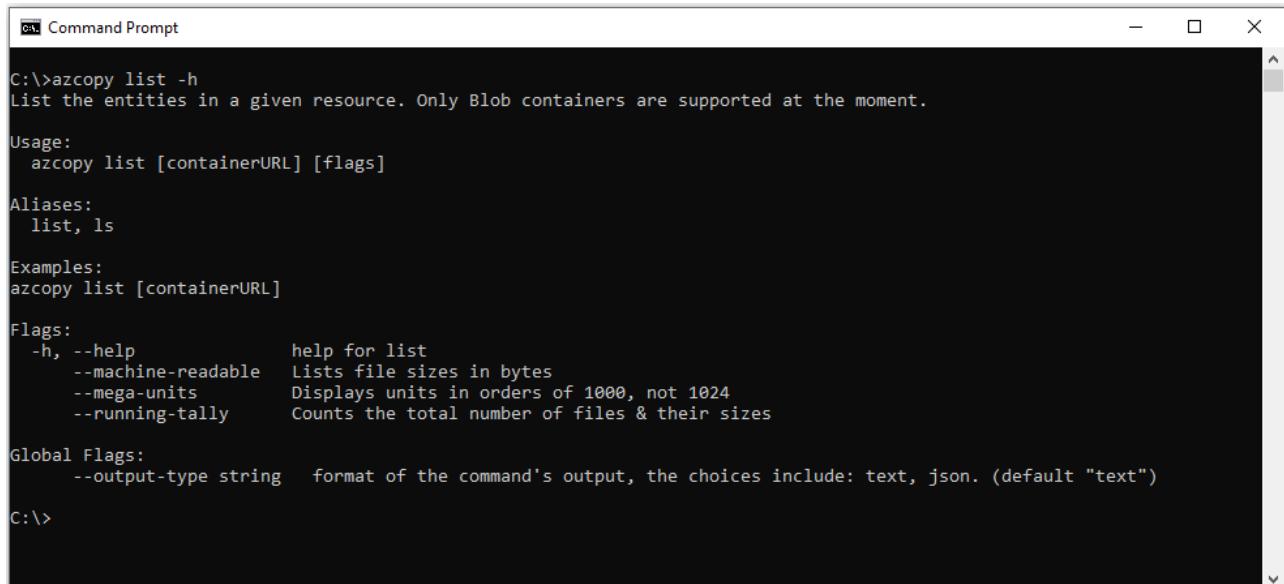
## Run AzCopy

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.

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`).



```
C:\>azcopy list -h
List the entities in a given resource. Only Blob containers are supported at the moment.

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 & their sizes

Global Flags:
 --output-type string format of the command's output, the choices include: text, json. (default "text")

C:\>
```

To find detailed reference documentation for each command and command parameter, see [azcopy](#)

**NOTE**

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.

## Choose how you'll provide authorization credentials

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

By using Azure Active Directory, you can provide credentials once instead of having to append a SAS token to each command.

**NOTE**

In the current release, if you plan to copy blobs between storage accounts, you'll have to append a SAS token to each source URL. You can omit the SAS token only from the destination URL. For examples, see [Copy blobs between storage accounts](#).

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](#) has been assigned to your user identity, managed identity, or service principal.

User identities, managed identities, and service principals are each a type of *security principal*, so we'll use the term *security principal* for the remainder of this article.

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 [Grant access to Azure blob and queue data with RBAC in the Azure portal](#).

**NOTE**

Keep in mind that RBAC 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 in Azure Data Lake Storage Gen2](#).

#### Authenticate 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.

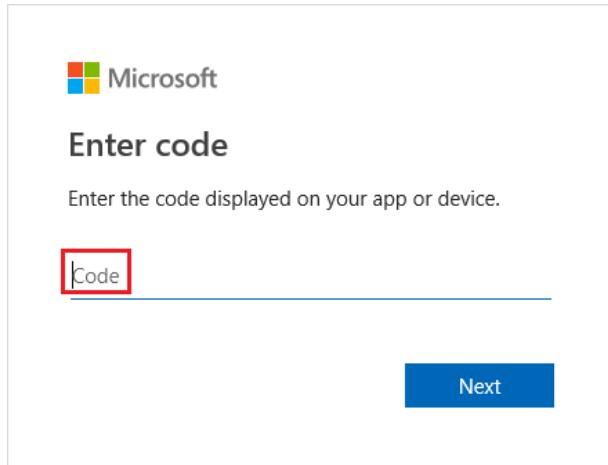
```
azcopy login
```

If you belong to more than one organization, include 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.

#### Authenticate 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 [Authenticate 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](#)

#### 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.

#### 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 login.

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.

#### Authenticate 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 the 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](#).

#### 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
```

#### 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.

#### 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 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 \(SAS\)](#).

## Transfer files

After you've authenticated your identity or obtained a SAS token, you can begin transferring files.

To find example commands, see any of these articles.

- [Transfer data with AzCopy and blob storage](#)
- [Transfer data with AzCopy and file storage](#)
- [Transfer data with AzCopy and Amazon S3 buckets](#)
- [Transfer data with AzCopy and Azure Stack storage](#)

## Use AzCopy 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 (un-changing) 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	<code>curl -v https://aka.ms/downloadazcopy-v10-linux</code>
Windows	<code>(curl https://aka.ms/downloadazcopy-v10-windows -MaximumRedirection 0 -ErrorAction silentlyContinue).RawContent</code>

#### 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	<pre>wget -O azcopy_v10.tar.gz https://aka.ms/downloadazcopy-v10- linux &amp;&amp; tar -xf azcopy_v10.tar.gz --strip-components=1</pre>
Windows	<pre>Invoke-WebRequest https://azcopyvnext.azureedge.net/release20190517/azcopy_windows_amd64_1 -OutFile azcopyv10.zip &lt;&lt;Unzip here&gt;&gt;</pre>

#### 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 AzCopy 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.

## Use the previous version of AzCopy

If you need to use the previous version of AzCopy, see either of the following links:

- [AzCopy on Windows \(v8\)](#)
- [AzCopy on Linux \(v7\)](#)

## Configure, optimize, and troubleshoot AzCopy

See [Configure, optimize, and troubleshoot AzCopy](#)

## Next steps

If you have questions, issues, or general feedback, submit them [on GitHub](#) page.

# Transfer data with AzCopy and Blob storage

4/10/2020 • 10 minutes to read • [Edit Online](#)

AzCopy is a command-line utility that you can use to copy data to, from, or between storage accounts. This article contains example commands that work with Blob storage.

## TIP

The examples in this article 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 ('').

## 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 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://<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. The examples in this section create a container named `mycontainer`.

Syntax	<code>azcopy make 'https://&lt;storage-account-name&gt;.&lt;blob or dfs&gt;.core.windows.net/&lt;container-name&gt;'</code>
Example	<code>azcopy make 'https://mystorageaccount.blob.core.windows.net/mycontainer'</code>
Example (hierarchical namespace)	<code>azcopy make 'https://mystorageaccount.dfs.core.windows.net/mycontainer'</code>

For detailed reference docs, see [azcopy make](#).

## Upload files

You can use the `azcopy copy` command to upload files and directories from your local computer.

This section contains the following examples:

- Upload a file
- Upload a directory
- Upload the contents of a directory
- Upload specific files

## TIP

You can tweak your upload operation by using optional flags. Here's a few examples.

### SCENARIO

Upload files as Append Blobs or Page Blobs.

### FLAG

--blob-type=[BlockBlob|PageBlob|AppendBlob]

Upload 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](#).

## Upload a file

### Syntax

```
azcopy copy '<local-file-path>' 'https://<storage-account-name>.blob.core.windows.net/<container-name>/<blob-name>'
```

### Example

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

### Example (hierarchical namespace)

```
azcopy copy 'C:\myDirectory\myTextFile.txt' 'https://mystorageaccount.dfs.core.windows.net/mycontainer/myTextFile.t'
```

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

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.

### Syntax

```
azcopy copy '<local-directory-path>' 'https://<storage-account-name>.blob.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/myBlobDirec --recursive
```

### Example (hierarchical namespace)

```
azcopy copy 'C:\myDirectory' 'https://mystorageaccount.dfs.core.windows.net/mycontainer/myBlobDirect --recursive
```

If you specify the name of a directory that does not exist in the container, AzCopy creates a new directory by that name.

## Upload the contents of a directory

You can upload the contents of a directory without copying the containing directory itself by using the wildcard symbol (\*).

## Syntax

```
azcopy copy '<local-directory-path>*' 'https://<storage-account-name>.blob.core.windows.net/<container-name>/<directory-path>'
```

## Example

```
azcopy copy 'C:\myDirectory*' 'https://mystorageaccount.blob.core.windows.net/mycontainer/myBlobDirec'
```

## Example (hierarchical namespace)

```
azcopy copy 'C:\myDirectory*' 'https://mystorageaccount.dfs.core.windows.net/mycontainer/myBlobDirecto'
```

### NOTE

Append the `--recursive` flag to upload files in all sub-directories.

## Upload specific files

You can specify complete file names, or use partial names with wildcard characters (\*).

### 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.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. You need to 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.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.

## Download files

You can use the [azcopy copy](#) command to download blobs, directories, and containers to your local computer.

This section contains the following examples:

- Download a file
- Download a directory
- Download the contents of a directory
- Download specific files

#### TIP

You can tweak your download operation by using optional flags. Here's a few examples.

SCENARIO	FLAG
Automatically decompress files.	--decompress=[gzip deflate]
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](#).

#### 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 file

Syntax	<pre>azcopy copy 'https://&lt;storage-account-name&gt;.&lt;blob or dfs&gt;.core.windows.net/&lt;container-name&gt;/&lt;blob-path&gt;' '&lt;local- file-path&gt;'</pre>
Example	<pre>azcopy copy 'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFile. 'C:\myDirectory\myTextFile.txt'</pre>
Example (hierarchical namespace)	<pre>azcopy copy 'https://mystorageaccount.dfs.core.windows.net/mycontainer/myTextFile.t 'C:\myDirectory\myTextFile.txt'</pre>

## Download a directory

Syntax	<pre>azcopy copy 'https://&lt;storage-account-name&gt;.&lt;blob or dfs&gt;.core.windows.net/&lt;container-name&gt;/&lt;directory-path&gt;' '&lt;local- directory-path&gt;' --recursive</pre>
Example	<pre>azcopy copy 'https://mystorageaccount.blob.core.windows.net/mycontainer/myBlobDirec 'C:\myDirectory' --recursive</pre>
Example (hierarchical namespace)	<pre>azcopy copy 'https://mystorageaccount.dfs.core.windows.net/mycontainer/myBlobDirect 'C:\myDirectory' --recursive</pre>

This example results in a directory named `C:\myDirectory\myBlobDirectory` that contains all of the downloaded files.

## Download the contents of a directory

You can download the contents of a directory without copying the containing directory itself by using the wildcard symbol (\*).

#### 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/myBlobDirec
'C:\myDirectory'
```

#### NOTE

Append the `--recursive` flag to download files in all sub-directories.

### Download specific files

You can specify complete file names, or use partial names with wildcard characters (\*).

#### 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 '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/FileDirecto
'C:\myDirectory' --include-path 'photos;documents\myFile.txt' --recursi
```

#### Example (hierarchical namespace)

```
azcopy copy
'https://mystorageaccount.dfs.core.windows.net/mycontainer/FileDirector
'C:\myDirectory' --include-path 'photos;documents\myFile.txt'--recursiv
```

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. You need to include the `--recursive` option to transfer all files in the `https://mystorageaccount.blob.core.windows.net/mycontainer/FileDirectory/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 '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/FileDirecto
'C:\myDirectory' --include-pattern 'myFile*.txt;*.pdf'
```

#### Example (hierarchical namespace)

```
azcopy copy
'https://mystorageaccount.dfs.core.windows.net/mycontainer/FileDirector
'C:\myDirectory' --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.

## Copy blobs between storage accounts

You can use AzCopy to copy blobs to other storage accounts. The copy operation is synchronous so when the command returns, that indicates that all files have been copied.

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. You can increase the throughput of these operations by setting the value of the `AZCOPY_CONCURRENCY_VALUE` environment variable. To learn more, see [Optimize throughput](#).

### NOTE

This scenario has the following limitations in the current release.

- You have to append a SAS token to each source URL. If you provide authorization credentials by using Azure Active Directory (AD), you can omit the SAS token only from the destination URL.
- Premium block blob storage accounts don't support access tiers. 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`).

This section contains the following examples:

- Copy a blob to another storage account
- Copy a directory to another storage account
- Copy a container to another storage account
- Copy all containers, directories, and files to another storage account

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.

### TIP

You can tweak your copy operation by using optional flags. Here's a few examples.

SCENARIO	FLAG
Copy files as Append Blobs or Page Blobs.	<code>--blob-type=[BlockBlob PageBlob AppendBlob]</code>
Copy to a specific access tier (such as the archive tier).	<code>--block-blob-tier=[None Hot Cool Archive]</code>
Automatically decompress files.	<code>--decompress=[gzip deflate]</code>

For a complete list, see [options](#).

### Copy a blob to another storage account

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

Syntax	<pre>azcopy copy 'https://&lt;source-storage-account-name&gt;.blob.core.windows.net/&lt;container-name&gt;/&lt;blob-path&gt;&lt;SAS-token&gt;' 'https://&lt;destination-storage-account-name&gt;.blob.core.windows.net/&lt;container-name&gt;/&lt;blob-path&gt;'</pre>
Example	<pre>azcopy copy 'https://mysourceaccount.blob.core.windows.net/mycontainer/myTextFile.txt?sv=2018-03-28&amp;ss=b&amp;st=2018-03-28T00:00:00Z&amp;se=2018-03-28T00:00:00Z&amp;sr=c&amp;sp=r&amp;sig=...&amp;api-version=2018-03-28' 'https://mydestinationaccount.blob.core.windows.net/mycontainer/myTextFile.txt?sv=2018-03-28&amp;ss=b&amp;st=2018-03-28T00:00:00Z&amp;se=2018-03-28T00:00:00Z&amp;sr=c&amp;sp=r&amp;sig=...&amp;api-version=2018-03-28'</pre>
Example (hierarchical namespace)	<pre>azcopy copy 'https://mysourceaccount.blob.core.windows.net/mycontainer/myTextFile.txt?sv=2018-03-28&amp;ss=b&amp;st=2018-03-28T00:00:00Z&amp;se=2018-03-28T00:00:00Z&amp;sr=c&amp;sp=r&amp;sig=...&amp;api-version=2018-03-28' 'https://mydestinationaccount.blob.core.windows.net/mycontainer/myTextFile.txt?sv=2018-03-28&amp;ss=b&amp;st=2018-03-28T00:00:00Z&amp;se=2018-03-28T00:00:00Z&amp;sr=c&amp;sp=r&amp;sig=...&amp;api-version=2018-03-28'</pre>

### Copy a directory to another storage account

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

#### Syntax

```
azcopy copy 'https://<source-storage-account-name>.blob.core.windows.net/<container-name>/<directory-path> <SAS-token>' 'https://<destination-storage-account-name>.blob.core.windows.net/<container-name>' --recursive
```

#### Example

```
azcopy copy
'https://mysourceaccount.blob.core.windows.net/mycontainer/myBlobDirect
sv=2018-03-28&ss=bfqt&srt=sco&sp=rw&lacup&se=2019-07-04T05:30:08Z&st=20
07-
03T21:30:08Z&spr=https&sig=CAfhgnc9gdGktvB=ska7bAiqIdM845yiyFwdMH481QA
'https://mydestinationaccount.blob.core.windows.net/mycontainer' --recu
```

#### Example (hierarchical namespace)

```
azcopy copy
'https://mysourceaccount.blob.core.windows.net/mycontainer/myBlobDirect
sv=2018-03-28&ss=bfqt&srt=sco&sp=rw&lacup&se=2019-07-04T05:30:08Z&st=20
07-
03T21:30:08Z&spr=https&sig=CAfhgnc9gdGktvB=ska7bAiqIdM845yiyFwdMH481QA
'https://mydestinationaccount.blob.core.windows.net/mycontainer' --recu
```

## Copy a container to another storage account

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

#### Syntax

```
azcopy copy '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 copy 'https://mysourceaccount.blob.core.windows.net/mycontainer?
sv=2018-03-28&ss=bfqt&srt=sco&sp=rw&lacup&se=2019-07-04T05:30:08Z&st=20
07-
03T21:30:08Z&spr=https&sig=CAfhgnc9gdGktvB=ska7bAiqIdM845yiyFwdMH481QA
'https://mydestinationaccount.blob.core.windows.net/mycontainer' --recu
```

#### Example (hierarchical namespace)

```
azcopy copy 'https://mysourceaccount.blob.core.windows.net/mycontainer?
sv=2018-03-28&ss=bfqt&srt=sco&sp=rw&lacup&se=2019-07-04T05:30:08Z&st=20
07-
03T21:30:08Z&spr=https&sig=CAfhgnc9gdGktvB=ska7bAiqIdM845yiyFwdMH481QA
'https://mydestinationaccount.blob.core.windows.net/mycontainer' --recu
```

## Copy all containers, directories, and blobs to another storage account

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

#### Syntax

```
azcopy copy 'https://<source-storage-account-name>.blob.core.windows.net/<SAS-token>'
'https://<destination-storage-account-name>.blob.core.windows.net/' --recursive
```

#### Example

```
azcopy copy 'https://mysourceaccount.blob.core.windows.net/?sv=2018-03-
28&ss=bfqt&srt=sco&sp=rw&lacup&se=2019-07-04T05:30:08Z&st=2019-07-
03T21:30:08Z&spr=https&sig=CAfhgnc9gdGktvB=ska7bAiqIdM845yiyFwdMH481QA
'https://mydestinationaccount.blob.core.windows.net' --recursive
```

#### Example (hierarchical namespace)

```
azcopy copy 'https://mysourceaccount.blob.core.windows.net/?sv=2018-03-
28&ss=bfqt&srt=sco&sp=rw&lacup&se=2019-07-04T05:30:08Z&st=2019-07-
03T21:30:08Z&spr=https&sig=CAfhgnc9gdGktvB=ska7bAiqIdM845yiyFwdMH481QA
'https://mydestinationaccount.blob.core.windows.net' --recursive
```

## Synchronize files

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).

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`.

#### NOTE

To prevent accidental deletions, make sure to enable the `soft delete` feature before you use the `--delete-destination=prompt|true` flag.

#### TIP

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.	<code>--check-md5=[NoCheck LogOnly FailIfDifferent FailIfDifferentOrMissing]</code>
Exclude files based on a pattern.	<code>--exclude-path</code>
Specify how detailed you want your sync-related log entries to be.	<code>--log-level=[WARNING ERROR INFO NONE]</code>

For a complete list, see [options](#).

### 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.

#### 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.

#### 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.

#### Syntax

```
azcopy sync '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 sync
'https://mysourceaccount.blob.core.windows.net/mycontainer'
'https://mydestinationaccount.blob.core.windows.net/mycontainer'
--recursive
```

## Update a directory with changes to a directory in another file share

The first directory that appears in this command is the source. The second one is the destination.

## Syntax

```
azcopy sync 'https://<source-storage-account-
name>.blob.core.windows.net/<container-name>/<directory-
name>' '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'
'https://mydestinationaccount.blob.core.windows.net/mycontainer/myDirec
--recursive
```

## Next steps

Find more examples in any of these articles:

- [Get started with AzCopy](#)
- [Tutorial: Migrate on-premises data to cloud storage by using AzCopy](#)
- [Transfer data with AzCopy and file storage](#)
- [Transfer data with AzCopy and Amazon S3 buckets](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

# Copy data from Amazon S3 to Azure Storage by using AzCopy

1/14/2020 • 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 the 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.

#### IMPORTANT

This feature is currently in preview. If you decide to remove data from your S3 buckets after a copy operation, make sure to verify that the data was properly copied to your storage account before you remove the data.

## 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'
```

### Example (hierarchical namespace)

```
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]]

(<https://docs.aws.amazon.com/AmazonS3/latest/dev/VirtualHosting.html>).

## 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/mydirec --recursive=true
```

### Example (hierarchical namespace)

```
azcopy copy 'https://s3.amazonaws.com/mybucket/mydirectory' 'https://mystorageaccount.blob.core.windows.net/mycontainer/mydirec --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
```

### Example (hierarchical namespace)

```
azcopy copy 'https://s3.amazonaws.com/mybucket/mydirectory'
'https://mystorageaccount.blob.core.windows.net/mycontainer/mydirec
--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
```

### Example (hierarchical namespace)

```
azcopy copy 'https://s3.amazonaws.com/mybucket/mydirectory'
'https://mystorageaccount.blob.core.windows.net/mycontainer/mydirec
--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
```

### Example (hierarchical namespace)

```
azcopy copy 'https://s3.amazonaws.com/mybucket/mydirectory'
'https://mystorageaccount.blob.core.windows.net/mycontainer/mydirec
--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-invalid-metadata-handle` 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
ExcludeIfInvalid	(Default option) The metadata isn't included in the transferred object. AzCopy logs a warning.
FailIfInvalid	Objects aren't copied. AzCopy logs an error and includes that error in the failed count that appears in the transfer summary.
RenameIfInvalid	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 and 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 any of these articles:

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and blob storage](#)
- [Transfer data with AzCopy and file storage](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

# Configure, optimize, and troubleshoot AzCopy

4/10/2020 • 7 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 perform advanced configuration tasks and helps you to troubleshoot issues that can arise as you use AzCopy.

## NOTE

If you're looking for content to help you get started with AzCopy, see any of the following 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](#)

## 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: <code>set https_proxy=&lt;proxy IP&gt;:&lt;proxy port&gt;</code> In PowerShell use: <code>\$env:https_proxy=&lt;proxy IP&gt;:&lt;proxy port&gt;"</code>
Linux	<code>export https_proxy=&lt;proxy IP&gt;:&lt;proxy port&gt;</code>
MacOS	<code>export https_proxy=&lt;proxy IP&gt;:&lt;proxy port&gt;</code>

Currently, AzCopy doesn't support proxies that require authentication with NTLM or Kerberos.

## Optimize performance

You can benchmark performance, and then use commands and environment variables to find an optimal tradeoff between performance and resource consumption.

This section helps you perform these optimization tasks:

- Run benchmark tests
- Optimize throughput
- Optimize memory use
- Optimize file synchronization

### Run benchmark tests

You can run a performance benchmark test on specific blob containers to view general performance statistics and to identify performance bottlenecks.

Use the following command to run a performance benchmark test.

Syntax	<code>azcopy bench 'https://&lt;storage-account-name&gt;.blob.core.windows.net/&lt;container-name&gt;'</code>
Example	<code>azcopy bench 'https://mystorageaccount.blob.core.windows.net/mycontainer/myBlobDirect ct sv=2018-03-28&amp;ss=bjqt&amp;srs=sco&amp;sp=rjkjhjup&amp;se=2019-05-10T04:37:48Z&amp;st=2019 09T20:37:48Z&amp;spr=https&amp;sig=%2FSOVEFfsKDqRry4bk3qz1vAQFwY5DDzp2%2B%2F3Eykf</code>

#### 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.

For detailed reference docs, see [azcopy bench](#).

To view detailed help guidance for this command, type `azcopy bench -h` and then press the ENTER key.

#### Optimize throughput

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
```

Throughput can decrease when transferring small files. 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 3000, 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.

#### Optimize memory use

Set the `AZCOPY_BUFFER_GB` environment variable to specify the maximum amount of your system memory you want AzCopy to use 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>

#### 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`.

## Troubleshoot issues

AzCopy creates log and plan files for every job. You can use the 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 if you want.

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
```

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>
azcopy jobs resume <job-id> --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 the plan and log files

By default, plan and log files are located in the `%USERPROFILE%\azcopy` directory on Windows, or in the `$HOME$\azcopy` directory on Mac and Linux. You can change this location.

### Change the location of plan files

Use any of these commands.

OPERATING SYSTEM	COMMAND
Windows	<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	<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: `NONE`, `DEBUG`, `INFO`, `WARNING`, `ERROR`, `PANIC`, and `FATAL`.

## 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.

# Copy and transform data in Azure Blob storage by using Azure Data Factory

4/15/2020 • 26 minutes to read • [Edit Online](#)

APPLIES TO: Azure Data Factory Azure Synapse Analytics (Preview)

This article outlines how to use Copy Activity in Azure Data Factory to copy data from and to Azure Blob storage, and use Data Flow to transform data in Azure Blob storage. To learn about Azure Data Factory, read the [introductory article](#).

## TIP

For data lake or data warehouse migration scenario, learn more from [Use Azure Data Factory to migrate data from your data lake or data warehouse to Azure](#).

## Supported capabilities

This Azure Blob connector is supported for the following activities:

- [Copy activity](#) with [supported source/sink matrix](#)
- [Mapping data flow](#)
- [Lookup activity](#)
- [GetMetadata activity](#)
- [Delete activity](#)

For 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 account key, service shared access signature, service principal or managed identities for Azure resources 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](#).
- [Preserve file metadata during copy](#).

## IMPORTANT

If you enable the [Allow trusted Microsoft services to access this storage account](#) option on Azure Storage firewall settings and want to use Azure integration runtime to connect to your Blob Storage, you must use [managed identity authentication](#).

## Get started

You can use one of the following tools or SDKs to use the copy activity with a pipeline. Select a link for step-by-step instructions:

- [Copy data tool](#)
- [Azure portal](#)
- [.NET SDK](#)

- [Python SDK](#)
- [Azure PowerShell](#)
- [REST API](#)
- [Azure Resource Manager template](#)

The following sections provide details about properties that are used to define Data Factory entities specific to Blob storage.

## Linked service properties

Azure Blob connector support the following authentication types, refer to the corresponding section on details:

- [Account key authentication](#)
- [Shared access signature authentication](#)
- [Service principal authentication](#)
- [Managed identities for Azure resources authentication](#)

### NOTE

When using PolyBase to load data into SQL Data Warehouse, if your source or staging Blob storage is configured with Virtual Network endpoint, you must use managed identity authentication as required by PolyBase, and use Self-hosted Integration Runtime with version 3.18 or above. See the [managed identity authentication](#) section with more configuration prerequisites.

### NOTE

HDInsights and Azure Machine Learning activities only support Azure Blob storage account key authentication.

### 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>AzureBlobStorage</b> (suggested) or <b>AzureStorage</b> (see notes below).	Yes
connectionString	Specify the information needed to connect to Storage for the connectionString property. You can also put account key in Azure Key Vault and pull the <code>accountKey</code> configuration out of the connection string. Refer to the following samples and <a href="#">Store credentials in Azure Key Vault</a> article with more details.	Yes
connectVia	The <a href="#">integration runtime</a> to be used to connect to the data store. You can use Azure Integration Runtime or Self-hosted Integration Runtime (if your data store is in a private network). If not specified, it uses the default Azure Integration Runtime.	No

**NOTE**

Secondary Blob Service Endpoint is not supported when using account key authentication. You can use other authentication types.

**NOTE**

If you were using "AzureStorage" type linked service, it is still supported as-is, while you are suggested to use this 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 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

- Data Factory 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 \(SAS\)](#).
- In later dataset configuration, the folder path is the absolute path starting from container level. You need to configure one aligned with the path in your SAS URI.

To use shared access signature authentication, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to <b>AzureBlobStorage</b> (suggested) or <b>AzureStorage</b> (see notes below).	Yes
sasUri	Specify the shared access signature URI to the Storage resources such as blob/container. Mark this field as a SecureString to store it securely in Data Factory. You can also put SAS token in Azure Key Vault to leverage auto rotation and remove the token portion. Refer to the following samples and <a href="#">Store credentials in Azure Key Vault</a> article with more details.	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 located in a private network). If not specified, it uses the default Azure Integration Runtime.	No

#### NOTE

If you were using "AzureStorage" type linked service, it is still supported as-is, while you are suggested to use this 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://<container>.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 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://<container>.blob.core.windows.net/>"
 },
 "sasToken": {
 "type": "AzureKeyVaultSecret",
 "store": {
 "referenceName": "<Azure Key Vault linked service name>",
 "type": "LinkedServiceReference"
 },
 "secretName": "<secretName>"
 }
 },
 "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 in your data factory.
- 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/blob based on the need. A shared access signature URI to a blob allows Data Factory to access that particular blob. A shared access signature URI to a Blob storage container allows Data Factory 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 Azure Storage service principal authentication in general, refer to [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 the following 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. Refer to [Manage access rights to Azure Storage data with RBAC](#) with more details on the roles.
  - As **source**, in Access control (IAM), grant at least **Storage Blob Data Reader** role.
  - As **sink**, in Access control (IAM), grant at least **Storage Blob Data Contributor** role.

These properties are supported for an Azure Blob storage linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type 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
servicePrincipalId	Specify the application's client ID.	Yes
servicePrincipalKey	Specify the application's key. Mark this field as a <b>SecureString</b> to store it securely in Data Factory, 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 the mouse in the top-right corner of the Azure portal.	Yes
connectVia	The <a href="#">integration runtime</a> to be used to connect to the data store. You can use Azure Integration Runtime or Self-hosted Integration Runtime (if your data store is in a private network). If not specified, it uses the default Azure Integration Runtime.	No

#### NOTE

Service principal authentication is only supported by "AzureBlobStorage" type linked service but not previous "AzureStorage" type linked service.

**Example:**

```
{
 "name": "AzureBlobStorageLinkedService",
 "properties": {
 "type": "AzureBlobStorage",
 "typeProperties": {
 "serviceEndpoint": "https://<accountName>.blob.core.windows.net/",
 "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"
 }
 }
}
```

## Managed identities for Azure resources authentication

A data factory can be associated with a [managed identity for Azure resources](#), which represents this specific data factory. You can directly use this managed identity for Blob storage authentication similar to using your own service principal. It allows this designated factory to access and copy data from/to your Blob storage.

Refer to [Authenticate access to Azure Storage using Azure Active Directory](#) for Azure Storage authentication in general. To use managed identities for Azure resources authentication, follow these steps:

1. [Retrieve data factory managed identity information](#) by copying the value of **managed identity object ID** generated along with your factory.
2. Grant the managed identity proper permission in Azure Blob storage. Refer to [Manage access rights to Azure Storage data with RBAC](#) with more details on the roles.
  - **As source**, in Access control (IAM), grant at least **Storage Blob Data Reader** role.
  - **As sink**, in Access control (IAM), grant at least **Storage Blob Data Contributor** role.

### IMPORTANT

If you use PolyBase to load data from Blob (as source or as staging) into SQL Data Warehouse, when using managed identity authentication for Blob, make sure you also follow steps 1 and 2 in [this guidance](#) to 1) register your SQL Database server with Azure Active Directory (Azure AD) and 2) assign the Storage Blob Data Contributor role to your SQL Database server; the rest are handled by Data Factory. If your Blob storage is configured with an Azure Virtual Network endpoint, to use PolyBase to load data from it, you must use managed identity authentication as required by PolyBase.

These properties are supported for an Azure Blob storage linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to <b>AzureBlobStorage</b> .	Yes
serviceEndpoint	Specify the Azure Blob storage service endpoint with the pattern of <div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">https://&lt;accountName&gt;.blob.core.windows.net/</div>	Yes

PROPERTY	DESCRIPTION	REQUIRED
connectVia	The <a href="#">integration runtime</a> to be used to connect to the data store. You can use Azure Integration Runtime or Self-hosted Integration Runtime (if your data store is in a private network). If not specified, it uses the default Azure Integration Runtime.	No

#### NOTE

Managed identities for Azure resources authentication is only supported by "AzureBlobStorage" type linked service but not previous "AzureStorage" type linked service.

#### Example:

```
{
 "name": "AzureBlobStorageLinkedService",
 "properties": {
 "type": "AzureBlobStorage",
 "typeProperties": {
 "serviceEndpoint": "https://<accountName>.blob.core.windows.net/"
 },
 "connectVia": {
 "referenceName": "<name of Integration Runtime>",
 "type": "IntegrationRuntimeReference"
 }
 }
}
```

## Dataset properties

For a full list of sections and properties available for defining datasets, see the [Datasets](#) article.

Azure Data Factory support the following file formats. Refer to each article on format-based settings.

- [Avro format](#)
- [Binary format](#)
- [Delimited text format](#)
- [JSON format](#)
- [ORC format](#)
- [Parquet format](#)

The following properties are supported for Azure Blob under `location` settings in format-based dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the location in dataset must be set to <a href="#">AzureBlobStorageLocation</a> .	Yes
container	The blob container.	Yes

PROPERTY	DESCRIPTION	REQUIRED
folderPath	The path to folder under the given container. If you want to use wildcard to filter folder, skip this setting and specify in activity source settings.	No
fileName	The file name under the given container + folderPath. If you want to use wildcard to filter files, skip this setting and specify 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 supported by the Blob storage source and sink.

### Blob storage as a source type

Azure Data Factory support the following file formats. Refer to each article on format-based settings.

- [Avro format](#)
- [Binary format](#)
- [Delimited text format](#)
- [JSON format](#)
- [ORC format](#)
- [Parquet format](#)

The following properties are supported for Azure Blob under `storeSettings` settings in format-based copy source:

PROPERTY	DESCRIPTION	REQUIRED
----------	-------------	----------

PROPERTY	DESCRIPTION	REQUIRED
type	The type property under <code>storeSettings</code> must be set to <code>AzureBlobStorageReadSettings</code> .	Yes
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 <code>true</code> (default) and <code>false</code> .	No
prefix	Prefix for the blob name under the given container configured in dataset to filter source blobs. Blobs whose name starts with this prefix are selected. Applies only when <code>wildcardFolderPath</code> and <code>wildcardFileName</code> properties are not specified.	No
wildcardFolderPath	The folder path with wildcard characters under the given container 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
wildcardFileName	The file name with wildcard characters under the given container + <code>folderPath/wildcardFolderPath</code> 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 folder name has wildcard or this escape char inside. See more examples in <a href="#">Folder and file filter examples</a> .	Yes if <code>fileName</code> is not specified in dataset

PROPERTY	DESCRIPTION	REQUIRED
modifiedDatetimeStart	<p>Files filter based on the attribute: Last Modified. The files will be selected if their last modified time are within the time range between <code>modifiedDatetimeStart</code> and <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 mean 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. 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>	No
modifiedDatetimeEnd	Same as above.	No
maxConcurrentConnections	The number of the connections to connect to storage store concurrently. Specify only when you want to limit the concurrent connection to the data store.	No

#### NOTE

For Parquet/delimited text format, **BlobSource** type copy activity source mentioned in next section is still supported as-is for backward compatibility. You are suggested to use this new model going forward, and the ADF 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>"
 }
 }
 }
]

```

## Blob storage as a sink type

Azure Data Factory support the following file formats. Refer to each article on format-based settings.

- [Avro format](#)
- [Binary format](#)
- [Delimited text format](#)
- [JSON format](#)
- [ORC format](#)
- [Parquet format](#)

The following properties are supported for Azure Blob under `storeSettings` settings in format-based copy sink:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property under <code>storeSettings</code> must be set to <code>AzureBlobStorageWriteSettings</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 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
blockSizeInMB	<p>Specify the block size in MB used to write data to block blobs. Learn more <a href="#">about Block Blobs</a>.</p> <p>Allowed value is <b>between 4 and 100 MB</b>.</p> <p>By default, ADF automatically determine the block size based on your source store type and data. For non-binary copy into Blob, the default block size is 100 MB so as to fit in at most 4.95 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 <math>\text{blockSizeInMB} * 50000</math> is big enough to store the data, otherwise copy activity run will fail.</p>	No
maxConcurrentConnections	The number of the connections to connect to storage store concurrently. Specify only when you want to limit the concurrent connection to the data store.	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"
 }
 }
 }
 }
]

```

## 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
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 File6.csv

FOLDERPATH	FILENAME	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
container/Folder*	*.csv	false	container FolderA <b>File1.csv</b> File2.json Subfolder1 File3.csv File4.json File5.csv AnotherFolderB File6.csv
container/Folder*	*.csv	true	container FolderA <b>File1.csv</b> File2.json Subfolder1 <b>File3.csv</b> File4.json <b>File5.csv</b> AnotherFolderB File6.csv

### 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 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

Recursive	CopyBehavior	Source Folder Structure	Resulting Target
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 folder Folder1 is created with the following structure:  Folder1 File1 File2  Subfolder1 with File3, File4, and File5 is not picked up.
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.

## 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](#).

## Mapping data flow properties

When transforming data in mapping data flow, you can read and write files from Azure Blob Storage in JSON, Avro, Delimited Text, or Parquet format. For more information, see [source transformation](#) and [sink transformation](#) in the

mapping data flow feature.

## Source transformation

In the source transformation, you can read from a container, folder or individual file in Azure Blob Storage. 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. The configuration includes:

- Wildcard paths:** mycontainer / partdata/\*\*/\*.csv
- Partition root path:** partdata
- List of files:** (checkbox)
- Multiline rows:** (checkbox)
- Column to store file name:** fileName
- After completion:** (radio buttons) No action (selected), Delete source files, Move
- Start time (UTC):** 02/01/2020 00:00:00
- End time (UTC):** 02/02/2020 00:00:00
- Filter by last modified:** (checkbox)

**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	* DELETE 0	* UPSERT 0	* 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 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.

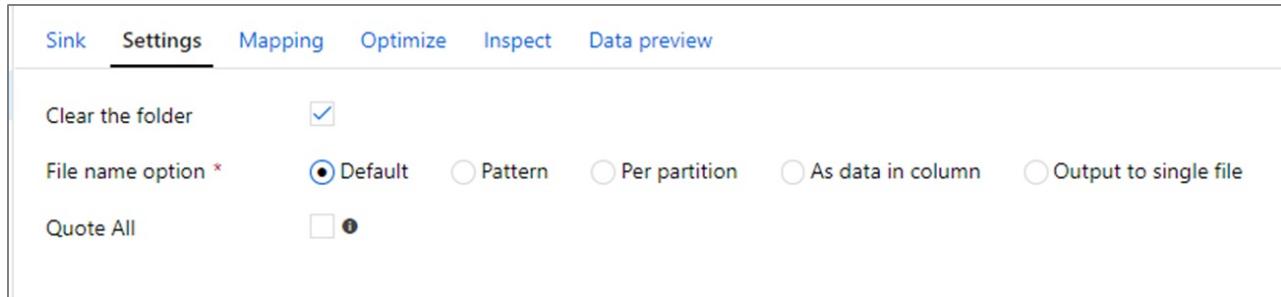
## NOTE

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.

## Sink properties

In the sink transformation, you can write to either a container or folder in Azure Blob Storage. the **Settings** tab lets you manage how the files get written.



**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

## 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
type	The type property of the dataset must be set to <b>AzureBlob</b> .	Yes
folderPath	<p>Path to the container and folder in the blob storage.</p> <p>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 actual folder name has wildcard or this escape char inside.</p> <p>Examples: myblobcontainer/myblobfolder/, see more examples in <a href="#">Folder and file filter examples</a>.</p>	Yes for Copy/Lookup activity, No for GetMetadata activity
fileName	<p><b>Name or wildcard filter</b> for the blob(s) under the specified "FolderPath". If you don't specify a value for this property, the dataset points to all blobs in the folder.</p> <p>For 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 actual file name has wildcard or this escape char 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 blob name with the following pattern: <code>"Data.[activity run ID GUID].[GUID if FlattenHierarchy].[format if configured].[compression if configured]"</code>, e.g. <code>"Data.0a405f8a-93ff-4c6f-b3be-f69616f1df7a.txt.gz"</code>; if you copy from tabular source using table name instead of query, the name pattern is <code>"[table name].[format].[compression if configured]"</code>, e.g. <code>"MyTable.csv"</code>.</p>	No

PROPERTY	DESCRIPTION	REQUIRED
modifiedDatetimeStart	<p>Files filter based on the attribute: Last Modified. The files will be selected if their last modified time are within the time range between <code>modifiedDatetimeStart</code> and <code>modifiedDatetimeEnd</code>. The time is applied to UTC time zone in the format of "2018-12-01T05:00:00Z".</p> <p>Be aware the overall performance of data movement will be impacted by enabling this setting when you want to do file filter from huge amounts of files.</p> <p>The properties can be NULL that mean 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. 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>	No
modifiedDatetimeEnd	<p>Files filter based on the attribute: Last Modified. The files will be selected if their last modified time are within the time range between <code>modifiedDatetimeStart</code> and <code>modifiedDatetimeEnd</code>. The time is applied to UTC time zone in the format of "2018-12-01T05:00:00Z".</p> <p>Be aware the overall performance of data movement will be impacted by enabling this setting when you want to do file filter from huge amounts of files.</p> <p>The properties can be NULL that mean 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. 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>	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 <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** with folder part and **fileName** with file name.

To copy a subset of blobs under a folder, specify **FolderPath** with folder part and **fileName** with 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 copy activity source model

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to <b>BlobSource</b> .	Yes
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> .	No
maxConcurrentConnections	The number of the connections to connect to storage store concurrently. Specify only when you want to limit the concurrent connection to the data store.	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 copy activity sink model

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity sink must be set to <b>BlobSink</b> .	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 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 number of the connections to connect to storage store concurrently. Specify only when you want to limit the concurrent connection to the data store.	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"
 }
 }
 }
]

```

## Next steps

For a list of data stores supported as sources and sinks by the copy activity in Data Factory, see [Supported data stores](#).

# How to mount Blob storage as a file system with blobfuse

3/20/2020 • 3 minutes to read • [Edit Online](#)

## Overview

**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, read the details in [the blobfuse repository](#).

### 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 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 supports installation on Ubuntu 14.04, 16.04, and 18.04. 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 6 distribution:

```
sudo rpm -Uvh https://packages.microsoft.com/config/rhel/6/packages-microsoft-prod.rpm
```

Similarly, change the URL to `.../rhel/7/...` to point to an Enterprise Linux 7 distribution.

Another example on an Ubuntu 14.04 distribution:

```
wget https://packages.microsoft.com/config/ubuntu/14.04/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
```

## 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 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. In Ubuntu distributions, this ephemeral disk is mounted on '/mnt'. In Red Hat and CentOS distributions, the disk is mounted on '/mnt/resource/'.

Make sure your user has access to the temporary path:

```
sudo mkdir /mnt/resource/blobfusetmp -p
sudo chown <youruser> /mnt/resource/blobfusetmp
```

### Configure your storage account credentials

Blobfuse requires your credentials to be stored in a text file in the following format:

```
accountName myaccount
accountKey storageaccesskey
containerName mycontainer
```

The `accountName` is the prefix for your storage account - not the full URL.

Create this file using:

```
touch ~/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 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'.

```
sudo 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
```

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
```

## Next steps

- [Blobfuse home page](#)
- [Report blobfuse issues](#)

# Transfer data with the Data Movement library

3/10/2020 • 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 .NET Core. 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  $8 *$  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, 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(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.

# Frequently asked questions about Azure Storage migration

4/5/2020 • 11 minutes to read • [Edit Online](#)

This article answers common questions about Azure Storage migration.

## Copy, upload or download

### How do I create a script to copy files from one container to another?

To copy files between containers, you can use AzCopy. See the following example:

```
AzCopy /Source:https://xxx.blob.core.windows.net/xxx
/Dest:https://xxx.blob.core.windows.net/xxx /SourceKey:xxx /DestKey:xxx
/S
```

AzCopy uses the [Copy Blob API](#) to copy each file in the container.

You can use any virtual machine or local machine that has internet access to run AzCopy. You can also use an Azure Batch schedule to do this automatically, but it's more complicated.

The automation script is designed for Azure Resource Manager deployment instead of storage content manipulation. For more information, see [Deploy resources with Resource Manager templates and Azure PowerShell](#).

### Is there any charge for copying data between two file shares on the same storage account within the same region?

No. There is no charge for this process.

### How can I download 1-2 TB of data from the Azure portal?

Use AzCopy to download the data. For more information, see [Transfer data with AzCopy on Windows](#) and [Transfer data with AzCopy on Linux](#).

### How can I download a VHD to a local machine, other than by using the download option in the portal?

You can use [Storage Explorer](#) to download a VHD.

### How do I download data to a Linux-based computer from an Azure storage account, or upload data from a Linux machine?

You can use the Azure CLI.

- Download a single blob:

```
azure storage blob download -k "<Account Key>" -a "<Storage Account Name>" --container "<Blob Container Name>" -b "<Remote File Name>" -d "<Local path where the file will be downloaded to>"
```

- Upload a single blob:

```
azure storage blob upload -k "<Account Key>" -a "<Storage Account Name>" --container "<Blob Container Name>" -f "<Local File Name>"
```

## How do I migrate Blobs from one storage account to another?

You can do this using our [Blob migration script](#).

## Migration or backup

### How do I move data from one storage container to another?

Follow these steps:

1. Create the container (folder) in the destination blob.
2. Use [AzCopy](#) to copy the contents from the original blob container to a different blob container.

### How do I create a PowerShell script to move data from one Azure file share to another in Azure Storage?

Use AzCopy to move the data from one Azure file share to another in Azure Storage. For more information, see [Transfer data with AzCopy on Windows](#) and [Transfer data with AzCopy on Linux](#).

### How do I upload large .csv files to Azure Storage?

Use AzCopy to upload large .csv files to Azure Storage. For more information, see [Transfer data with AzCopy on Windows](#) and [Transfer data with AzCopy on Linux](#).

### I have to move the logs from drive D to my Azure storage account every day. How do I automate this?

You can use AzCopy and create a task in Task Scheduler. Upload files to an Azure storage account by using an AzCopy batch script. For more information, see [How to configure and run startup tasks for a cloud service](#).

### How do I move my storage account between subscriptions?

Use AzCopy to move your storage account between subscriptions. For more information, see [Transfer data with AzCopy on Windows](#) and [Transfer data with AzCopy on Linux](#).

### How can I move about 10 TB of data to storage at another region?

Use AzCopy to move the data. For more information, see [Transfer data with AzCopy on Windows](#) and [Transfer data with AzCopy on Linux](#).

### How can I copy data from on-premises to Azure Storage?

Use AzCopy to copy the data. For more information, see [Transfer data with AzCopy on Windows](#) and [Transfer data with AzCopy on Linux](#).

### How can I move data from on-premises to Azure Files?

Use AzCopy to move data. For more information, see [Transfer data with AzCopy on Windows](#) and [Transfer data with AzCopy on Linux](#).

### How do I move managed disks to another storage account?

## NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

Follow these steps:

1. Stop the virtual machine that the managed disk is attached to.
2. Copy the managed disk VHD from one area to another by running the following Azure PowerShell script:

```
Connect-AzAccount

Select-AzSubscription -SubscriptionId <ID>

$sas = Grant-AzDiskAccess -ResourceGroupName <RG name> -DiskName <Disk name> -DurationInSecond 3600 -
Access Read

$destContext = New-AzStorageContext -StorageAccountName contosostorageav1 -StorageAccountKey <your
account key>

Start-AzStorageBlobCopy -AbsoluteUri $sas.AccessSAS -DestContainer 'vhds' -DestContext $destContext -
DestBlob 'MyDestinationBlobName.vhd'
```

3. Create a managed disk by using the VHD file in another region to which you copied the VHD. To do this, run the following Azure PowerShell script:

```
$resourceGroupName = 'MDDemo'

$diskName = 'contoso\os_disk1'

$vhdUri = 'https://contoso.storageaccou.com.vhd

$storageId = '/subscriptions/<ID>/resourceGroups/<RG
name>/providers/Microsoft.Storage/storageAccounts/contosostorageaccount1'

$location = 'westus'

$storageType = 'StandardLRS'

$diskConfig = New-AzDiskConfig -AccountType $storageType -Location $location -CreateOption Import -
SourceUri $vhdUri -StorageAccountId $storageId -DiskSizeGB 128

$osDisk = New-AzDisk -DiskName $diskName -Disk $diskConfig -ResourceGroupName $resourceGroupName
```

For more information about how to deploy a virtual machine from a managed disk, see [CreateVmFromManagedOsDisk.ps1](#).

### How do I move or download data from a storage account?

Use AzCopy to download the data. For more information, see [Transfer data with AzCopy on Windows](#) and [Transfer data with AzCopy on Linux](#).

### How do I move from a premium storage account to a standard storage account?

Follow these steps:

1. Create a standard storage account. (Or use an existing standard storage account in your subscription.)

## 2. Download AzCopy. Run one of the following AzCopy commands.

To copy whole disks in the storage account:

```
AzCopy /Source:https://sourceaccount.blob.core.windows.net/mycontainer1
/Dest:https://destaccount.blob.core.windows.net/mycontainer2
/SourceKey:key1 /DestKey:key2 /S
```

To copy only one disk, provide the name of the disk in **Pattern**:

```
AzCopy /Source:https://sourceaccount.blob.core.windows.net/mycontainer1
/Dest:https://destaccount.blob.core.windows.net/mycontainer2
/SourceKey:key1 /DestKey:key2 /Pattern:abc.vhd
```

The operation might take several hours to finish.

To make sure that the transfer finished successfully, examine the destination storage account container in the Azure portal. After the disks are copied to the standard storage account, you can attach them to the virtual machine as an existing disk. For more information, see [How to attach a managed data disk to a Windows virtual machine in the Azure portal](#).

## How do I move from a classic storage account to an Azure Resource Manager storage account?

You can use the **Move-AzureStorageAccount** cmdlet. This cmdlet has multiple steps (validate, prepare, commit). You can validate the move before you make it.

If you have virtual machines, you must take additional steps before you migrate the storage account data. For more information, see [Migrate IaaS resources from classic to Azure Resource Manager by using Azure PowerShell](#).

## How do I back up my entire storage account to another storage account?

There is no option to back up an entire storage account directly. But you can manually move the container in that storage account to another account by using AzCopy or Storage Explorer. The following steps show how to use AzCopy to move the container:

1. Install the [AzCopy](#) command-line tool. This tool helps you move the VHD file between storage accounts.
2. After you install AzCopy on Windows by using the installer, open a Command Prompt window and then browse to the AzCopy installation folder on your computer. By default, AzCopy is installed to **%ProgramFiles(x86)%\Microsoft SDKs\Azure\AzCopy** or **%ProgramFiles%\Microsoft SDKs\Azure\AzCopy**.
3. Run the following command to move the container. You must replace the text with the actual values.

```
AzCopy /Source:https://sourceaccount.blob.core.windows.net/mycontainer1
/Dest:https://destaccount.blob.core.windows.net/mycontainer2
/SourceKey:key1 /DestKey:key2 /S
```

- **/Source** : Provide the URI for the source storage account (up to the container).
- **/Dest** : Provide the URI for the target storage account (up to the container).
- **/SourceKey** : Provide the primary key for the source storage account. You can copy this key from the Azure portal by selecting the storage account.
- **/DestKey** : Provide the primary key for the target storage account. You can copy this key from the portal by selecting the storage account.

After you run this command, the container files are moved to the target storage account.

#### **NOTE**

The AzCopy CLI does not work together with the **Pattern** switch when you copy from one Azure blob to another.

You can directly copy and edit the AzCopy command, and cross-check to make sure that **Pattern** matches the source. Also make sure that **/S** wildcards are in effect. For more information, see [AzCopy parameters](#).

## How do I back up Azure file storage?

There is no backup solution. However, Azure Files also supports asynchronous copy. So, you can copy files:

- From a share to another share within a storage account or to a different storage account.
- From a share to a blob container within a storage account or to a different storage account.

For more information, see [Transfer data with AzCopy on Windows](#).

## Configuration

### How do I change the secondary location to the Europe region for a storage account?

When you create a storage account, you select the primary region for the account. The selection of the secondary region is based on the primary region, and it cannot be changed. For more information, see [Geo-redundant storage \(GRS\): Cross-regional replication for Azure Storage](#).

### Where can I get more information about Azure Storage Service Encryption (SSE)?

See the following articles:

- [Azure Storage security guide](#)
- [Azure Storage Service Encryption for Data at Rest](#)

### How do I encrypt data in a storage account?

After you enable encryption in a storage account, the existing data is not encrypted. To encrypt the existing data, you must upload it again to the storage account.

Use AzCopy to copy the data to a different storage account and then move the data back. You can also use [encryption at rest](#).

### Are there any prerequisites for changing the replication of a storage account from geo-redundant storage to locally redundant storage?

No.

### How do I convert to Azure Premium Storage for a file share?

Premium Storage is not allowed on an Azure file share.

### How do I upgrade from a standard storage account to a premium storage account? How do I downgrade from a premium storage account to a standard storage account?

You must create the destination storage account, copy data from the source account to the destination account, and then delete the source account. You can use a tool such as AzCopy to copy the data.

If you have virtual machines, you must take additional steps before you migrate the storage account data. For more information, see [Migrating to Azure Premium Storage \(unmanaged disks\)](#).

### How can I give other people access to my storage resources?

To give other people access to the storage resources:

- Use a shared access signature (SAS) token to provide access to a resource.
- Provide a user with the primary or secondary key for the storage account. For more information, see [Manage storage account access keys](#).
- Change the access policy to allow anonymous access. For more information, see [Grant anonymous users permissions to containers and blobs](#).

**Where is AzCopy installed?**

- If you access AzCopy from the Microsoft Azure Storage command line, type **AzCopy**. The command line is installed alongside AzCopy.
- If you installed the 32-bit version, it's located here: **%ProgramFiles(x86)%\Microsoft SDKs\Azure\AzCopy**.
- If you installed the 64-bit version, it's located here: **%ProgramFiles%\Microsoft SDKs\Azure\AzCopy**.

**How do I use an HTTPS custom domain with my storage account? For example, how do I make "https://mystorageaccountname.blob.core.windows.net/images/image.gif" appear as "https://www.contoso.com/images/image.gif"?**

TLS/SSL is not currently supported on storage accounts with custom domains. But you can use non-HTTPS custom domains. For more information, see [Configure a custom domain name for your Blob storage endpoint](#).

## Access to storage

**How do I map a container folder on a virtual machine?**

Use an Azure file share.

**How do I access Azure Files redundant storage?**

Read-access geo-redundant storage is required to access redundant storage. However, Azure Files supports only locally redundant storage and standard geo-redundant storage that does not allow read-only access.

**For a replicated storage account (such as zone-redundant storage, geo-redundant storage, or read-access geo-redundant storage), how do I access data that is stored in the secondary region?**

- If you're using zone-redundant storage or geo-redundant storage, you cannot access data from the secondary region unless you initiate a failover to that region. For more information about the failover process, see [Disaster recovery and storage account failover \(preview\) in Azure Storage](#).
- If you're using read-access geo-redundant storage, you can access data from the secondary region at any time. Use one of the following methods:
  - **AzCopy**: Append **-secondary** to the storage account name in the URL to access the secondary endpoint. For example:  
<https://storageaccountname-secondary.blob.core.windows.net/vhds/BlobName.vhd>
  - **SAS token**: Use an SAS token to access data from the endpoint. For more information, see [Using shared access signatures](#).

**How do I use FTP to access data that is in a storage account?**

There is no way to access a storage account directly by using FTP. However, you can set up an Azure virtual machine, and then install an FTP server on the virtual machine. You can have the FTP server store files on an Azure Files share or on a data disk that is available to the virtual machine.

If you want only to download data without having to use Storage Explorer or a similar application, you might be able to use an SAS token. For more information, see [Using shared access signatures](#).

## Need help? Contact support.

If you still need help, [contact support](#) to get your issue resolved quickly.

# How to use Blob storage from C++

1/14/2020 • 8 minutes to read • [Edit Online](#)

This guide demonstrates how to perform common scenarios using Azure Blob storage. The examples show how to upload, list, download, and delete blobs. The samples are written in C++ and use the [Azure Storage Client Library for C++](#).

To learn more about Blob storage, see [Introduction to Azure Blob storage](#).

## NOTE

This guide targets the Azure Storage Client Library for C++ version 1.0.0 and above. Microsoft recommends using the latest version of the Storage Client Library for C++, available via [NuGet](#) or [GitHub](#).

## Create an Azure storage account

The easiest way to create your first Azure storage account is by using the [Azure portal](#). To learn more, see [Create a storage account](#).

You can also create an Azure storage account by using [Azure PowerShell](#), [Azure CLI](#), or the [Azure Storage Resource Provider for .NET](#).

If you prefer not to create a storage account in Azure at this time, you can also use the Azure storage emulator to run and test your code in a local environment. For more information, see [Use the Azure Storage Emulator for Development and Testing](#).

## Create a C++ application

In this guide, you will use storage features which can be run within a C++ application.

To do so, you will need to install the Azure Storage Client Library for C++ and create an Azure storage account in your Azure subscription.

To install the Azure Storage Client Library for C++, you can use the following methods:

- **Linux:** Follow the instructions given in the [Azure Storage Client Library for C++ README: Getting Started on Linux](#) page.
- **Windows:** On Windows, use `vcpkg` as the dependency manager. Follow the [quick-start](#) to initialize vcpkg. Then, use the following command to install the library:

```
.\vcpkg.exe install azure-storage-cpp
```

You can find a guide for how to build the source code and export to NuGet in the [README](#) file.

## Configure your application to access Blob storage

Add the following include statements to the top of the C++ file where you want to use the Azure storage APIs to access blobs:

```
#include <was/storage_account.h>
#include <was/blob.h>
#include <cpprest/filestream.h>
#include <cpprest/containerstream.h>
```

## Setup an Azure storage connection string

An Azure storage client uses a storage connection string to store endpoints and credentials for accessing data management services. When running in a client application, you must provide the storage connection string in the following format, using the name of your storage account and the storage access key for the storage account listed in the [Azure Portal](#) for the *AccountName* and *AccountKey* values. For information on storage accounts and access keys, see [About Azure Storage Accounts](#). This example shows how you can declare a static field to hold the connection string:

```
// Define the connection-string with your values.
const utility::string_t
storage_connection_string(U("DefaultEndpointsProtocol=https;AccountName=your_storage_account;AccountKey=your_st
orage_account_key"));
```

To test your application in your local Windows computer, you can use the Microsoft Azure [storage emulator](#) that is installed with the [Azure SDK](#). The storage emulator is a utility that simulates the Blob, Queue, and Table services available in Azure on your local development machine. The following example shows how you can declare a static field to hold the connection string to your local storage emulator:

```
// Define the connection-string with Azure Storage Emulator.
const utility::string_t storage_connection_string(U("UseDevelopmentStorage=true;"));
```

To start the Azure storage emulator, Select the **Start** button or press the **Windows** key. Begin typing **Azure Storage Emulator**, and select **Microsoft Azure Storage Emulator** from the list of applications.

The following samples assume that you have used one of these two methods to get the storage connection string.

## Retrieve your storage account

You can use the **cloud\_storage\_account** class to represent your Storage Account information. To retrieve your storage account information from the storage connection string, you can use the **parse** method.

```
// Retrieve storage account from connection string.
azure::storage::cloud_storage_account storage_account =
azure::storage::cloud_storage_account::parse(storage_connection_string);
```

Next, get a reference to a **cloud\_blob\_client** class as it allows you to retrieve objects that represent containers and blobs stored within Blob storage. The following code creates a **cloud\_blob\_client** object using the storage account object we retrieved above:

```
// Create the blob client.
azure::storage::cloud_blob_client blob_client = storage_account.create_cloud_blob_client();
```

## How to: Create a container

Every blob in Azure storage must reside in a container. The container forms part of the blob name. For example, **mycontainer** is the name of the container in these sample blob URLs:

```
https://storagesample.blob.core.windows.net/mycontainer/blob1.txt
https://storagesample.blob.core.windows.net/mycontainer/photos/myphoto.jpg
```

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

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

#### IMPORTANT

Note that the name of a container must always be lowercase. If you include an upper-case letter in a container name, or otherwise violate the container naming rules, you may receive a 400 error (Bad Request).

This example shows how to create a container if it does not already exist:

```
try
{
 // Retrieve storage account from connection string.
 azure::storage::cloud_storage_account storage_account =
 azure::storage::cloud_storage_account::parse(storage_connection_string);

 // Create the blob client.
 azure::storage::cloud_blob_client blob_client = storage_account.create_cloud_blob_client();

 // Retrieve a reference to a container.
 azure::storage::cloud_blob_container container = blob_client.get_container_reference(U("my-sample-
container"));

 // Create the container if it doesn't already exist.
 container.create_if_not_exists();
}
catch (const std::exception& e)
{
 std::wcout << U("Error: ") << e.what() << std::endl;
}
```

By default, the new container is private and you must specify your storage access key to download blobs from this container. If you want to make the files (blobs) within the container available to everyone, you can set the container to be public using the following code:

```
// Make the blob container publicly accessible.
azure::storage::blob_container_permissions permissions;
permissions.set_public_access(azure::storage::blob_container_public_access_type::blob);
container.upload_permissions(permissions);
```

Anyone on the Internet can see blobs in a public container, but you can modify or delete them only if you have the appropriate access key.

## How to: Upload a blob into a container

Azure Blob storage supports block blobs and page blobs. In the majority of cases, block blob is the recommended type to use.

To upload a file to a block blob, get a container reference and use it to get a block blob reference. Once you have a blob reference, you can upload any stream of data to it by calling the **upload\_from\_stream** method. This operation will create the blob if it didn't previously exist, or overwrite it if it does exist. The following example shows how to upload a blob into a container and assumes that the container was already created.

```
// Retrieve storage account from connection string.
azure::storage::cloud_storage_account storage_account =
azure::storage::cloud_storage_account::parse(storage_connection_string);

// Create the blob client.
azure::storage::cloud_blob_client blob_client = storage_account.create_cloud_blob_client();

// Retrieve a reference to a previously created container.
azure::storage::cloud_blob_container container = blob_client.get_container_reference(U("my-sample-container"));

// Retrieve reference to a blob named "my-blob-1".
azure::storage::cloud_block_blob blockBlob = container.get_block_blob_reference(U("my-blob-1"));

// Create or overwrite the "my-blob-1" blob with contents from a local file.
concurrency::streams::istream input_stream =
concurrency::streams::file_stream<uint8_t>::open_istream(U("DataFile.txt")).get();
blockBlob.upload_from_stream(input_stream);
input_stream.close().wait();

// Create or overwrite the "my-blob-2" and "my-blob-3" blobs with contents from text.
// Retrieve a reference to a blob named "my-blob-2".
azure::storage::cloud_block_blob blob2 = container.get_block_blob_reference(U("my-blob-2"));
blob2.upload_text(U("more text"));

// Retrieve a reference to a blob named "my-blob-3".
azure::storage::cloud_block_blob blob3 = container.get_block_blob_reference(U("my-directory/my-sub-
directory/my-blob-3"));
blob3.upload_text(U("other text"));
```

Alternatively, you can use the **upload\_from\_file** method to upload a file to a block blob.

## How to: List the blobs in a container

To list the blobs in a container, first get a container reference. You can then use the container's **list\_blobs** method to retrieve the blobs and/or directories within it. To access the rich set of properties and methods for a returned **list\_blob\_item**, you must call the **list\_blob\_item.as\_blob** method to get a **cloud\_blob** object, or the **list\_blob.as\_directory** method to get a **cloud\_blob\_directory** object. The following code demonstrates how to retrieve and output the URI of each item in the **my-sample-container** container:

```

// Retrieve storage account from connection string.
azure::storage::cloud_storage_account storage_account =
azure::storage::cloud_storage_account::parse(storage_connection_string);

// Create the blob client.
azure::storage::cloud_blob_client blob_client = storage_account.create_cloud_blob_client();

// Retrieve a reference to a previously created container.
azure::storage::cloud_blob_container container = blob_client.get_container_reference(U("my-sample-container"));

// Output URI of each item.
azure::storage::list_blob_item_iterator end_of_results;
for (auto it = container.list_blobs(); it != end_of_results; ++it)
{
 if (it->is_blob())
 {
 std::wcout << U("Blob: ") << it->as_blob().uri().primary_uri().to_string() << std::endl;
 }
 else
 {
 std::wcout << U("Directory: ") << it->as_directory().uri().primary_uri().to_string() << std::endl;
 }
}

```

For more details on listing operations, see [List Azure Storage Resources in C++](#).

## How to: Download blobs

To download blobs, first retrieve a blob reference and then call the **download\_to\_stream** method. The following example uses the **download\_to\_stream** method to transfer the blob contents to a stream object that you can then persist to a local file.

```

// Retrieve storage account from connection string.
azure::storage::cloud_storage_account storage_account =
azure::storage::cloud_storage_account::parse(storage_connection_string);

// Create the blob client.
azure::storage::cloud_blob_client blob_client = storage_account.create_cloud_blob_client();

// Retrieve a reference to a previously created container.
azure::storage::cloud_blob_container container = blob_client.get_container_reference(U("my-sample-container"));

// Retrieve reference to a blob named "my-blob-1".
azure::storage::cloud_block_blob blockBlob = container.get_block_blob_reference(U("my-blob-1"));

// Save blob contents to a file.
concurrency::streams::container_buffer<std::vector<uint8_t>> buffer;
concurrency::streams::ostream output_stream(buffer);
blockBlob.download_to_stream(output_stream);

std::ofstream outfile("DownloadBlobFile.txt", std::ofstream::binary);
std::vector<unsigned char>& data = buffer.collection();

outfile.write((char *)&data[0], buffer.size());
outfile.close();

```

Alternatively, you can use the **download\_to\_file** method to download the contents of a blob to a file. In addition, you can also use the **download\_text** method to download the contents of a blob as a text string.

```
// Retrieve storage account from connection string.
azure::storage::cloud_storage_account storage_account =
azure::storage::cloud_storage_account::parse(storage_connection_string);

// Create the blob client.
azure::storage::cloud_blob_client blob_client = storage_account.create_cloud_blob_client();

// Retrieve a reference to a previously created container.
azure::storage::cloud_blob_container container = blob_client.get_container_reference(U("my-sample-container"));

// Retrieve reference to a blob named "my-blob-2".
azure::storage::cloud_block_blob text_blob = container.get_block_blob_reference(U("my-blob-2"));

// Download the contents of a blob as a text string.
utility::string_t text = text_blob.download_text();
```

## How to: Delete blobs

To delete a blob, first get a blob reference and then call the `delete_blob` method on it.

```
// Retrieve storage account from connection string.
azure::storage::cloud_storage_account storage_account =
azure::storage::cloud_storage_account::parse(storage_connection_string);

// Create the blob client.
azure::storage::cloud_blob_client blob_client = storage_account.create_cloud_blob_client();

// Retrieve a reference to a previously created container.
azure::storage::cloud_blob_container container = blob_client.get_container_reference(U("my-sample-container"));

// Retrieve reference to a blob named "my-blob-1".
azure::storage::cloud_block_blob blockBlob = container.get_block_blob_reference(U("my-blob-1"));

// Delete the blob.
blockBlob.delete_blob();
```

## Next steps

Now that you've learned the basics of blob storage, follow these links to learn more about Azure Storage.

- [How to use Queue Storage from C++](#)
- [How to use Table Storage from C++](#)
- [List Azure Storage Resources in C++](#)
- [Storage Client Library for C++ Reference](#)
- [Azure Storage Documentation](#)
- [Transfer data with the AzCopy command-line utility](#)

# How to use Blob storage from iOS

8/1/2019 • 12 minutes to read • [Edit Online](#)

This article shows how to perform common scenarios using Microsoft Azure Blob storage. The samples are written in Objective-C and use the [Azure Storage Client Library for iOS](#). The scenarios covered include uploading, listing, downloading, and deleting blobs. For more information on blobs, see the [Next Steps](#) section. You can also download the [sample app](#) to quickly see the use of Azure Storage in an iOS application.

To learn more about Blob storage, see [Introduction to Azure Blob storage](#).

## Create an Azure storage account

The easiest way to create your first Azure storage account is by using the [Azure portal](#). To learn more, see [Create a storage account](#).

You can also create an Azure storage account by using [Azure PowerShell](#), [Azure CLI](#), or the [Azure Storage Resource Provider for .NET](#).

If you prefer not to create a storage account in Azure at this time, you can also use the Azure storage emulator to run and test your code in a local environment. For more information, see [Use the Azure Storage Emulator for Development and Testing](#).

## Import the Azure Storage iOS library into your application

You can import the Azure Storage iOS library into your application either by using the [Azure Storage CocoaPod](#) or by importing the [Framework](#) file. CocoaPod is the recommended way as it makes integrating the library easier, however importing from the framework file is less intrusive for your existing project.

To use this library, you need the following:

- iOS 8+
- Xcode 7+

### CocoaPod

1. If you haven't done so already, [Install CocoaPods](#) on your computer by opening a terminal window and running the following command

```
sudo gem install cocoapods
```

2. Next, in the project directory (the directory containing your .xcodeproj file), create a new file called *Podfile*(no file extension). Add the following to *Podfile* and save.

```
platform :ios, '8.0'

target 'TargetName' do
 pod 'AZSClient'
end
```

3. In the terminal window, navigate to the project directory and run the following command

```
pod install
```

4. If your .xcodeproj is open in Xcode, close it. In your project directory open the newly created project file which will

have the .xcworkspace extension. This is the file you'll work from for now on.

## Framework

The other way to use the library is to build the framework manually:

1. First, download or clone the [azure-storage-ios repo](#).
2. Go into *azure-storage-ios -> Lib -> Azure Storage Client Library*, and open `AZSClient.xcodeproj` in Xcode.
3. At the top-left of Xcode, change the active scheme from "Azure Storage Client Library" to "Framework".
4. Build the project ( $\mathcal{B}$ +B). This will create an `AZSClient.framework` file on your Desktop.

You can then import the framework file into your application by doing the following:

1. Create a new project or open up your existing project in Xcode.
2. Drag and drop the `AZSClient.framework` into your Xcode project navigator.
3. Select *Copy items if needed*, and click on *Finish*.
4. Click on your project in the left-hand navigation and click the *General* tab at the top of the project editor.
5. Under the *Linked Frameworks and Libraries* section, click the Add button (+).
6. In the list of libraries already provided, search for `libxml2.2.tbd` and add it to your project.

## Import the Library

```
// Include the following import statement to use blob APIs.
#import <AZSClient/AZSClient.h>
```

If you are using Swift, you will need to create a bridging header and import `<AZSClient/AZSClient.h>` there:

1. Create a header file `Bridging-Header.h`, and add the above import statement.
2. Go to the *Build Settings* tab, and search for *Objective-C Bridging Header*.
3. Double-click on the field of *Objective-C Bridging Header* and add the path to your header file:  
`ProjectName/Bridging-Header.h`
4. Build the project ( $\mathcal{B}$ +B) to verify that the bridging header was picked up by Xcode.
5. Start using the library directly in any Swift file, there is no need for import statements.

## Configure your application to access Azure Storage

There are two ways to authenticate your application to access Storage services:

- Shared Key: Use Shared Key for testing purposes only
- Shared Access Signature (SAS): Use SAS for production applications

### Shared Key

Shared Key authentication means that your application will use your account name and account key to access Storage services. For the purposes of quickly showing how to use this library, we will be using Shared Key authentication in this getting started.

#### WARNING

Only use Shared Key authentication for testing purposes! Your account name and account key, which give full read/write access to the associated Storage account, will be distributed to every person that downloads your app. This is **not** a good practice as you risk having your key compromised by untrusted clients.

When using Shared Key authentication, you will create a [connection string](#). The connection string is comprised of:

- The **DefaultEndpointsProtocol** - you can choose HTTP or HTTPS. However, using HTTPS is highly recommended.
- The **Account Name** - the name of your storage account

- The **Account Key** - On the [Azure Portal](#), navigate to your storage account and click the **Keys** icon to find this information.
- (Optional) **EndpointSuffix** - This is used for storage services in regions with different endpoint suffixes, such as Azure China or Azure Governance.

Here is an example of connection string using Shared Key authentication:

```
"DefaultEndpointsProtocol=https;AccountName=your_account_name_here;AccountKey=your_account_key_here"
```

### Shared Access Signatures (SAS)

For a mobile application, the recommended method for authenticating a request by a client against the Azure Storage service is by using a Shared Access Signature (SAS). SAS allows you to grant a client access to a resource for a specified period of time, with a specified set of permissions. As the storage account owner, you'll need to generate a SAS for your mobile clients to consume. To generate the SAS, you'll probably want to write a separate service that generates the SAS to be distributed to your clients. For testing purposes, you can use the [Microsoft Azure Storage Explorer](#) or the [Azure Portal](#) to generate a SAS. When you create the SAS, you can specify the time interval over which the SAS is valid, and the permissions that the SAS grants to the client.

The following example shows how to use the Microsoft Azure Storage Explorer to generate a SAS.

1. If you haven't already, [Install the Microsoft Azure Storage Explorer](#)
2. Connect to your subscription.
3. Click on your Storage account and click on the "Actions" tab at the bottom left. Click "Get Shared Access Signature" to generate a "connection string" for your SAS.
4. Here is an example of a SAS connection string that grants read and write permissions at the service, container and object level for the blob service of the Storage account.

```
"SharedAccessSignature=sv=2015-04-05&ss=b&srt=sco&sp=rw&se=2016-07-21T18%3A00%3A00Z&sig=3ABdLOJZosCp00491T%2BqZGKIhafF1n1M3MzESDD3Gg%3D;BlobEndpoint=https://youraccount.blob.core.windows.net"
```

As you can see, when using a SAS, you're not exposing your account key in your application. You can learn more about SAS and best practices for using SAS by checking out [Shared Access Signatures: Understanding the SAS model](#).

## Asynchronous Operations

### NOTE

All methods that perform a request against the service are asynchronous operations. In the code samples, you'll find that these methods have a completion handler. Code inside the completion handler will run **after** the request is completed. Code after the completion handler will run **while** the request is being made.

## Create a container

Every blob in Azure Storage must reside in a container. The following example shows how to create a container, called *newcontainer*, in your Storage account if it doesn't already exist. When choosing a name for your container, be mindful of the naming rules mentioned above.

```

-(void)createContainer{
 NSError *accountCreationError;

 // Create a storage account object from a connection string.
 AZSCloudStorageAccount *account = [AZSCloudStorageAccount
accountFromConnectionString:@"DefaultEndpointsProtocol=https;AccountName=your_account_name_here;AccountKey=your_account_key_here" error:&accountCreationError];

 if(accountCreationError){
 NSLog(@"Error in creating account.");
 }

 // Create a blob service client object.
 AZSCloudBlobClient *blobClient = [account getBlobClient];

 // Create a local container object.
 AZSCloudBlobContainer *blobContainer = [blobClient containerReferenceFromName:@"newcontainer"];

 // Create container in your Storage account if the container doesn't already exist
 [blobContainer createContainerIfNotExistsWithCompletionHandler:^(NSError *error, BOOL exists) {
 if (error){
 NSLog(@"Error in creating container.");
 }
 }];
}

```

You can confirm that this works by looking at the [Microsoft Azure Storage Explorer](#) and verifying that *newcontainer* is in the list of containers for your Storage account.

## Set Container Permissions

A container's permissions are configured for **Private** access by default. However, containers provide a few different options for container access:

- **Private**: Container and blob data can be read by the account owner only.
- **Blob**: Blob data within this container can be read via anonymous request, but container data is not available. Clients cannot enumerate blobs within the container via anonymous request.
- **Container**: Container and blob data can be read via anonymous request. Clients can enumerate blobs within the container via anonymous request, but cannot enumerate containers within the storage account.

The following example shows you how to create a container with **Container** access permissions, which will allow public, read-only access for all users on the Internet:

```

-(void)createContainerWithPublicAccess{
 NSError *accountCreationError;

 // Create a storage account object from a connection string.
 AZSCloudStorageAccount *account = [AZSCloudStorageAccount
accountFromConnectionString:@"DefaultEndpointsProtocol=https;AccountName=your_account_name_here;AccountKey=your_account_key_here" error:&accountCreationError];

 if(accountCreationError){
 NSLog(@"Error in creating account.");
 }

 // Create a blob service client object.
 AZSCloudBlobClient *blobClient = [account getBlobClient];

 // Create a local container object.
 AZSCloudBlobContainer *blobContainer = [blobClient containerReferenceFromName:@"containerpublic"];

 // Create container in your Storage account if the container doesn't already exist
 [blobContainer createContainerIfNotExistsWithAccessType:AZSContainerPublicAccessTypeContainer requestOptions:nil
operationContext:nil completionHandler:^(NSError *error, BOOL exists){
 if (error){
 NSLog(@"Error in creating container.");
 }
 }];
}

```

## Upload a blob into a container

As mentioned in the Blob service concepts section, Blob Storage offers three different types of blobs: block blobs, append blobs, and page blobs. The Azure Storage iOS library supports all three types of blobs. In most cases, block blob is the recommended type to use.

The following example shows how to upload a block blob from an NSString. If a blob with the same name already exists in this container, the contents of this blob will be overwritten.

```

-(void)uploadBlobToContainer{
 NSError *accountCreationError;

 // Create a storage account object from a connection string.
 AZSCloudStorageAccount *account = [AZSCloudStorageAccount
accountFromConnectionString:@"DefaultEndpointsProtocol=https;AccountName=your_account_name_here;AccountKey=your_account_key_here" error:&accountCreationError];

 if(accountCreationError){
 NSLog(@"Error in creating account.");
 }

 // Create a blob service client object.
 AZSCloudBlobClient *blobClient = [account getBlobClient];

 // Create a local container object.
 AZSCloudBlobContainer *blobContainer = [blobClient containerReferenceFromName:@"containerpublic"];

 [blobContainer createContainerIfNotExistsWithAccessType:AZSContainerPublicAccessTypeContainer requestOptions:nil
operationContext:nil completionHandler:^(NSError *error, BOOL exists)
{
 if (error){
 NSLog(@"Error in creating container.");
 }
 else{
 // Create a local blob object
 AZSCloudBlockBlob *blockBlob = [blobContainer blockBlobReferenceFromName:@"sampleblob"];

 // Upload blob to Storage
 [blockBlob uploadFromText:@"This text will be uploaded to Blob Storage." completionHandler:^(NSError
*error) {
 if (error){
 NSLog(@"Error in creating blob.");
 }
 }];
 }
}];
}
}

```

You can confirm that this works by looking at the [Microsoft Azure Storage Explorer](#) and verifying that the container, *containerpublic*, contains the blob, *sampleblob*. In this sample, we used a public container so you can also verify that this application worked by going to the blobs URI:

```
https://nameofyourstorageaccount.blob.core.windows.net/containerpublic/sampleblob
```

In addition to uploading a block blob from an `NSString`, similar methods exist for `NSData`, `NSInputStream`, or a local file.

## List the blobs in a container

The following example shows how to list all blobs in a container. When performing this operation, be mindful of the following parameters:

- **continuationToken** - The continuation token represents where the listing operation should start. If no token is provided, it will list blobs from the beginning. Any number of blobs can be listed, from zero up to a set maximum. Even if this method returns zero results, if `results.continuationToken` is not nil, there may be more blobs on the service that have not been listed.
- **prefix** - You can specify the prefix to use for blob listing. Only blobs that begin with this prefix will be listed.
- **useFlatBlobListing** - As mentioned in the [Naming and referencing containers and blobs](#) section, although the Blob service is a flat storage scheme, you can create a virtual hierarchy by naming blobs with path information. However, non-flat listing is currently not supported. This feature is coming soon. For now, this value should be YES.
- **blobListingDetails** - You can specify which items to include when listing blobs
  - `AZSBlobListingDetailsNone`: List only committed blobs, and do not return blob metadata.

- *AZSBlobListingDetailsSnapshots*: List committed blobs and blob snapshots.
  - *AZSBlobListingDetailsMetadata*: Retrieve blob metadata for each blob returned in the listing.
  - *AZSBlobListingDetailsUncommittedBlobs*: List committed and uncommitted blobs.
  - *AZSBlobListingDetailsCopy*: Include copy properties in the listing.
  - *AZSBlobListingDetailsAll*: List all available committed blobs, uncommitted blobs, and snapshots, and return all metadata and copy status for those blobs.
- **maxResults** - The maximum number of results to return for this operation. Use -1 to not set a limit.
  - **completionHandler** - The block of code to execute with the results of the listing operation.

In this example, a helper method is used to recursively call the list blobs method every time a continuation token is returned.

```

-(void)listBlobsInContainer{
 NSError *accountCreationError;

 // Create a storage account object from a connection string.
 AZSCloudStorageAccount *account = [AZSCloudStorageAccount
accountFromConnectionString:@"DefaultEndpointsProtocol=https;AccountName=your_account_name_here;AccountKey=your_account_key_here" error:&accountCreationError];

 if(accountCreationError){
 NSLog(@"Error in creating account.");
 }

 // Create a blob service client object.
 AZSCloudBlobClient *blobClient = [account getBlobClient];

 // Create a local container object.
 AZSCloudBlobContainer *blobContainer = [blobClient containerReferenceFromName:@"containerpublic"];

 //List all blobs in container
 [self listBlobsInContainerHelper:blobContainer continuationToken:nil prefix:nil
blobListingDetails:AZSBlobListingDetailsAll maxResults:-1 completionHandler:^(NSError *error) {
 if (error != nil){
 NSLog(@"Error in creating container.");
 }
 }];
}

//List blobs helper method
-(void)listBlobsInContainerHelper:(AZSCloudBlobContainer *)container continuationToken:(AZSContinuationToken
*)continuationToken prefix:(NSString *)prefix blobListingDetails:(AZSBlobListingDetails)blobListingDetails
maxResults:(NSUInteger)maxResults completionHandler:(void (^)(NSError *))completionHandler
{
 [container listBlobsSegmentedWithContinuationToken:continuationToken prefix:prefix useFlatBlobListing:YES
blobListingDetails:blobListingDetails maxResults:maxResults completionHandler:^(NSError *error, AZSBlobResultSegment
*results) {
 if (error)
 {
 completionHandler(error);
 }
 else
 {
 for (int i = 0; i < results.blobs.count; i++) {
 NSLog(@"%@",[(AZSCloudBlockBlob *)results.blobs[i] blobName]);
 }
 if (results.continuationToken)
 {
 [self listBlobsInContainerHelper:container continuationToken:results.continuationToken prefix:prefix
blobListingDetails/blobListingDetails maxResults:maxResults completionHandler:completionHandler];
 }
 else
 {
 completionHandler(nil);
 }
 }
 }];
}
}

```

## Download a blob

The following example shows how to download a blob to a NSString object.

```

-(void)downloadBlobToString{
 NSError *accountCreationError;

 // Create a storage account object from a connection string.
 AZSCloudStorageAccount *account = [AZSCloudStorageAccount
accountFromConnectionString:@"DefaultEndpointsProtocol=https;AccountName=your_account_name_here;AccountKey=your_account_key_here" error:&accountCreationError];

 if(accountCreationError){
 NSLog(@"Error in creating account.");
 }

 // Create a blob service client object.
 AZSCloudBlobClient *blobClient = [account getBlobClient];

 // Create a local container object.
 AZSCloudBlobContainer *blobContainer = [blobClient containerReferenceFromName:@"containerpublic"];

 // Create a local blob object
 AZSCloudBlockBlob *blockBlob = [blobContainer blockBlobReferenceFromName:@"sampleblob"];

 // Download blob
 [blockBlob downloadToTextWithCompletionHandler:^(NSError *error, NSString *text) {
 if (error) {
 NSLog(@"Error in downloading blob");
 }
 else{
 NSLog(@"%@",text);
 }
 }];
}
}

```

## Delete a blob

The following example shows how to delete a blob.

```

-(void)deleteBlob{
 NSError *accountCreationError;

 // Create a storage account object from a connection string.
 AZSCloudStorageAccount *account = [AZSCloudStorageAccount
accountFromConnectionString:@"DefaultEndpointsProtocol=https;AccountName=your_account_name_here;AccountKey=your_account_key_here" error:&accountCreationError];

 if(accountCreationError){
 NSLog(@"Error in creating account.");
 }

 // Create a blob service client object.
 AZSCloudBlobClient *blobClient = [account getBlobClient];

 // Create a local container object.
 AZSCloudBlobContainer *blobContainer = [blobClient containerReferenceFromName:@"containerpublic"];

 // Create a local blob object
 AZSCloudBlockBlob *blockBlob = [blobContainer blockBlobReferenceFromName:@"sampleblob1"];

 // Delete blob
 [blockBlob deleteWithCompletionHandler:^(NSError *error) {
 if (error) {
 NSLog(@"Error in deleting blob.");
 }
 }];
}
}

```

## Delete a blob container

The following example shows how to delete a container.

```
- (void)deleteContainer{
 NSError *accountCreationError;

 // Create a storage account object from a connection string.
 AZSCloudStorageAccount *account = [AZSCloudStorageAccount
accountFromConnectionString:@"DefaultEndpointsProtocol=https;AccountName=your_account_name_here;AccountKey=your_account_key_here" error:&accountCreationError];

 if(accountCreationError){
 NSLog(@"Error in creating account.");
 }

 // Create a blob service client object.
 AZSCloudBlobClient *blobClient = [account getBlobClient];

 // Create a local container object.
 AZSCloudBlobContainer *blobContainer = [blobClient containerReferenceFromName:@"containerpublic"];

 // Delete container
 [blobContainer deleteContainerIfExistsWithCompletionHandler:^(NSError *error, BOOL success) {
 if(error){
 NSLog(@"Error in deleting container");
 }
 }];
}
```

## Next steps

Now that you've learned how to use Blob Storage from iOS, follow these links to learn more about the iOS library and the Storage service.

- [Azure Storage Client Library for iOS](#)
- [Azure Storage iOS Reference Documentation](#)
- [Azure Storage Services REST API](#)
- [Azure Storage Team Blog](#)

If you have questions regarding this library, feel free to post to our [MSDN Azure forum](#) or [Stack Overflow](#). If you have feature suggestions for Azure Storage, please post to [Azure Storage Feedback](#).

# How to use Blob Storage from Xamarin

8/1/2019 • 8 minutes to read • [Edit Online](#)

Xamarin enables developers to use a shared C# codebase to create iOS, Android, and Windows Store apps with their native user interfaces. This tutorial shows you how to use Azure Blob storage with a Xamarin application. If you'd like to learn more about Azure Storage, before diving into the code, see [Introduction to Microsoft Azure Storage](#).

## Create an Azure storage account

The easiest way to create your first Azure storage account is by using the [Azure portal](#). To learn more, see [Create a storage account](#).

You can also create an Azure storage account by using [Azure PowerShell](#), [Azure CLI](#), or the [Azure Storage Resource Provider for .NET](#).

If you prefer not to create a storage account in Azure at this time, you can also use the Azure storage emulator to run and test your code in a local environment. For more information, see [Use the Azure Storage Emulator for Development and Testing](#).

## Configure your application to access Azure Storage

There are two ways to authenticate your application to access Storage services:

- Shared Key: Use Shared Key for testing purposes only
- Shared Access Signature (SAS): Use SAS for production applications

### Shared Key

Shared Key authentication means that your application will use your account name and account key to access Storage services. For the purposes of quickly showing how to use this library, we will be using Shared Key authentication in this getting started.

#### WARNING

Only use Shared Key authentication for testing purposes! Your account name and account key, which give full read/write access to the associated Storage account, will be distributed to every person that downloads your app. This is **not** a good practice as you risk having your key compromised by untrusted clients.

When using Shared Key authentication, you will create a [connection string](#). The connection string is comprised of:

- The **DefaultEndpointsProtocol** - you can choose HTTP or HTTPS. However, using HTTPS is highly recommended.
- The **Account Name** - the name of your storage account
- The **Account Key** - On the [Azure Portal](#), navigate to your storage account and click the **Keys** icon to find this information.
- (Optional) **EndpointSuffix** - This is used for storage services in regions with different endpoint suffixes, such as Azure China or Azure Governance.

Here is an example of connection string using Shared Key authentication:

```
"DefaultEndpointsProtocol=https;AccountName=your_account_name_here;AccountKey=your_account_key_here"
```

### Shared Access Signatures (SAS)

For a mobile application, the recommended method for authenticating a request by a client against the Azure Storage service is by using a Shared Access Signature (SAS). SAS allows you to grant a client access to a resource for a specified period of time, with a specified set of permissions. As the storage account owner, you'll need to generate a SAS for your

mobile clients to consume. To generate the SAS, you'll probably want to write a separate service that generates the SAS to be distributed to your clients. For testing purposes, you can use the [Microsoft Azure Storage Explorer](#) or the [Azure Portal](#) to generate a SAS. When you create the SAS, you can specify the time interval over which the SAS is valid, and the permissions that the SAS grants to the client.

The following example shows how to use the Microsoft Azure Storage Explorer to generate a SAS.

1. If you haven't already, [Install the Microsoft Azure Storage Explorer](#)
2. Connect to your subscription.
3. Click on your Storage account and click on the "Actions" tab at the bottom left. Click "Get Shared Access Signature" to generate a "connection string" for your SAS.
4. Here is an example of a SAS connection string that grants read and write permissions at the service, container and object level for the blob service of the Storage account.

```
"SharedAccessSignature=sv=2015-04-05&ss=b&srt=sco&sp=rw&se=2016-07-21T18%3A00%3A00Z&sig=3ABdLOJZosCp0o491T%2BqZGKIhaffF1n1M3MzESDDD3Gg%3D;BlobEndpoint=https://youraccount.blob.core.windows.net"
```

As you can see, when using a SAS, you're not exposing your account key in your application. You can learn more about SAS and best practices for using SAS by checking out [Shared Access Signatures: Understanding the SAS model](#).

## Create a new Xamarin Application

For this tutorial, we'll be creating an app that targets Android, iOS, and Windows. This app will simply create a container and upload a blob into this container. We'll be using Visual Studio on Windows, but the same learnings can be applied when creating an app using Xamarin Studio on macOS.

Follow these steps to create your application:

1. If you haven't already, download and install [Xamarin for Visual Studio](#).
2. Open Visual Studio, and create a Blank App (Native Portable): **File > New > Project > Cross-Platform > Blank App(Native Portable)**.
3. Right-click your solution in the Solution Explorer pane and select **Manage NuGet Packages for Solution**. Search for **WindowsAzure.Storage** and install the latest stable version to all projects in your solution.
4. Build and run your project.

You should now have an application that allows you to click a button which increments a counter.

## Create container and upload blob

Next, under your `(Portable)` project, you'll add some code to `MyClass.cs`. This code creates a container and uploads a blob into this container. `MyClass.cs` should look like the following:

```

using Microsoft.WindowsAzure.Storage;
using Microsoft.WindowsAzure.Storage.Blob;
using System.Threading.Tasks;

namespace XamarinApp
{
 public class MyClass
 {
 public MyClass ()
 {

 }

 public static async Task performBlobOperation()
 {
 // Retrieve storage account from connection string.
 CloudStorageAccount storageAccount =
CloudStorageAccount.Parse("DefaultEndpointsProtocol=https;AccountName=your_account_name_here;AccountKey=your_account_key_here");

 // Create the blob client.
 CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

 // Retrieve reference to a previously created container.
 CloudBlobContainer container = blobClient.GetContainerReference("mycontainer");

 // Create the container if it doesn't already exist.
 await container.CreateIfNotExistsAsync();

 // Retrieve reference to a blob named "myblob".
 CloudBlockBlob blockBlob = container.GetBlockBlobReference("myblob");

 // Create the "myblob" blob with the text "Hello, world!"
 await blockBlob.UploadTextAsync("Hello, world!");
 }
 }
}

```

Make sure to replace "your\_account\_name\_here" and "your\_account\_key\_here" with your actual account name and account key.

Your iOS, Android, and Windows Phone projects all have references to your Portable project - meaning you can write all of your shared code in one place and use it across all of your projects. You can now add the following line of code to each project to start taking advantage: `MyClass.performBlobOperation()`

**XamarinApp.Droid > MainActivity.cs**

```
using Android.App;
using Android.Widget;
using Android.OS;

namespace XamarinApp.Droid
{
 [Activity (Label = "XamarinApp.Droid", MainLauncher = true, Icon = "@drawable/icon")]
 public class MainActivity : Activity
 {
 int count = 1;

 protected override async void OnCreate (Bundle bundle)
 {
 base.OnCreate (bundle);

 // Set our view from the "main" layout resource
 SetContentView (Resource.Layout.Main);

 // Get our button from the layout resource,
 // and attach an event to it
 Button button = FindViewById<Button> (Resource.Id.myButton);

 button.Click += delegate {
 button.Text = string.Format ("{0} clicks!", count++);
 };

 await MyClass.performBlobOperation();
 }
 }
}
```

## XamarinApp.iOS > ViewController.cs

```
using System;
using UIKit;

namespace XamarinApp.iOS
{
 public partial class ViewController : UIViewController
 {
 int count = 1;

 public ViewController (IntPtr handle) : base (handle)
 {
 }

 public override async void ViewDidLoad ()
 {
 int count = 1;

 public ViewController (IntPtr handle) : base (handle)
 {
 }

 public override async void ViewDidLoad ()
 {
 base.ViewDidLoad ();
 // Perform any additional setup after loading the view, typically from a nib.
 Button.AccessibilityIdentifier = "myButton";
 Button.TouchUpInside += delegate {
 var title = string.Format ("{0} clicks!", count++);
 ButtonSetTitle (title, UIControlState.Normal);
 };
 }

 await MyClass.performBlobOperation();
 }

 public override void DidReceiveMemoryWarning ()
 {
 base.DidReceiveMemoryWarning ();
 // Release any cached data, images, etc. that aren't in use.
 }
 }
}
```

XamarinApp.WinPhone > MainPage.xaml > MainPage.xaml.cs

```

using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Navigation;

// The Blank Page item template is documented at https://go.microsoft.com/fwlink/?LinkId=391641

namespace XamarinApp.WinPhone
{
 /// <summary>
 /// An empty page that can be used on its own or navigated to within a Frame.
 /// </summary>
 public sealed partial class MainPage : Page
 {
 int count = 1;

 public MainPage()
 {
 this.InitializeComponent();

 this.NavigationCacheMode = NavigationCacheMode.Required;
 }

 /// <summary>
 /// Invoked when this page is about to be displayed in a Frame.
 /// </summary>
 /// <param name="e">Event data that describes how this page was reached.
 /// This parameter is typically used to configure the page.</param>
 protected override async void OnNavigatedTo(NavigationEventArgs e)
 {
 int count = 1;

 public MainPage()
 {
 this.InitializeComponent();

 this.NavigationCacheMode = NavigationCacheMode.Required;
 }

 /// <summary>
 /// Invoked when this page is about to be displayed in a Frame.
 /// </summary>
 /// <param name="e">Event data that describes how this page was reached.
 /// This parameter is typically used to configure the page.</param>
 protected override async void OnNavigatedTo(NavigationEventArgs e)
 {
 // TODO: Prepare page for display here.

 // TODO: If your application contains multiple pages, ensure that you are
 // handling the hardware Back button by registering for the
 // Windows.Phone.UI.Input.HardwareButtons.BackPressed event.
 // If you are using the NavigationHelper provided by some templates,
 // this event is handled for you.
 Button.Click += delegate {
 var title = string.Format("{0} clicks!", count++);
 Button.Content = title;
 };

 await MyClass.performBlobOperation();
 }
 }
 }
}

```

## Run the application

You can now run this application in an Android or Windows Phone emulator. You can also run this application in an iOS emulator, but this will require a Mac. For specific instructions on how to do this, please read the documentation for [connecting Visual Studio to a Mac](#)

Once you run your app, it will create the container `mycontainer` in your Storage account. It should contain the blob, `myblob`, which has the text, `Hello, world!`. You can verify this by using the [Microsoft Azure Storage Explorer](#).

## Next steps

In this tutorial, you learned how to create a cross-platform application in Xamarin that uses Azure Storage, specifically focusing on one scenario in Blob Storage. However, you can do a lot more with not only Blob Storage, but also with Table, File, and Queue Storage. Please check out the following articles to learn more:

- [Get started with Azure Blob storage using .NET](#)
- [Introduction to Azure Files](#)
- [Develop for Azure Files with .NET](#)
- [Get started with Azure Table storage using .NET](#)
- [Get started with Azure Queue storage using .NET](#)

### 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.

# Move an Azure Storage account to another region

4/16/2020 • 6 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 whitelisted 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 > **Settings** > **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.

### 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

Here's some ways to move your data over.

#### ✓ Azure Storage Explorer

It's easy-to-use, and suitable for small data sets. You can copy containers and file shares, and then paste them into the target account.

See [Azure Storage Explorer](#):

#### ✓ AzCopy

This is the preferred approach. 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](#)

#### ✓ Azure Data Factory

Use this tool only if you need functionality that isn't supported in the current release of AzCopy. For example, in the current release of AzCopy, you can't copy blobs between accounts that have a hierarchical namespace. Also AzCopy doesn't preserve file access control lists or file timestamps (For example: create and modified time stamps).

See 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 File Storage 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](#)

# Troubleshoot latency using Storage Analytics logs

11/19/2019 • 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.

Logs   Out-Of-Band											
GetBlob											
And operation-type contains "GetBlob"											
<input checked="" type="checkbox"/> Filter											
<input checked="" type="checkbox"/> And operation-type contains "GetBlob"											
<input checked="" type="checkbox"/> Clear All											
version-number		operator-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-id
1.0	2019-10-07T19:17.33.781119.472	GetBlobProperties	Success	200	104	12	authenticated	contosoang	contosoang	blob	https://contosoang.blob.core.windows.net
1.0	2019-10-07T19:17.34.496027.712	GetBlobProperties	Success	200	106	26	authenticated	contosoang	contosoang	blob	https://contosoang.blob.core.windows.net
1.0	2019-10-07T19:17.34.496027.712	GetBlobProperties	Success	200	5	5	authenticated	contosoang	contosoang	blob	https://contosoang.blob.core.windows.net
1.0	2019-10-07T19:17.29.510242	GetBlobLeaseProperties	Success	200	9	9	authenticated	contosoang	contosoang	blob	https://contosoang.blob.core.windows.net
1.0	2019-10-07T19:20.35.550130.900	GetBlob	SASDeniedError	206	8401	181	contosoang	blob	contosoang	blob	https://contosoang.blob.core.windows.net
1.0	2019-10-07T19:20.35.550130.902	GetBlob	SASDeniedError	206	8457	515	contosoang	blob	contosoang	blob	https://contosoang.blob.core.windows.net
1.0	2019-10-07T19:20.35.550130.902	GetBlob	SASSuccess	200	10	10	sas	contosoang	contosoang	blob	https://contosoang.blob.core.windows.net
1.0	2019-10-07T19:20.35.550130.902	GetBlob	SASNeverExpire	206	8453	452	contosoang	blob	contosoang	blob	https://contosoang.blob.core.windows.net
1.0	2019-10-07T19:20.47.499914.742	GetBlobProperties	SASSuccess	200	10	10	sas	contosoang	contosoang	blob	https://contosoang.blob.core.windows.net
1.0	2019-10-07T19:20.47.499914.742	GetBlobProperties	SASNeverExpire	206	8454	344	sas	contosoang	contosoang	blob	https://contosoang.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 = 8062ms

The following table provides information about the high latency OperationType and RequestStatus results:

	REQUESTSTATUS=SUCCESS	REQUESTSTATUS=(SAS)NETWORKERROR	RECOMMENDATION
GetBlob	Yes	No	<a href="#">GetBlob Operation: RequestStatus = Success</a>

	REQUESTSTATUS= SUCCESS	REQUESTSTATUS= (SAS)NETWORKERROR	RECOMMENDATION
GetBlob	No	Yes	<a href="#">GetBlob Operation: RequestStatus = (SAS)NetworkError</a>
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.

# Scalability and performance targets for Blob storage

1/8/2020 • 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.

## 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	100 MiB
Maximum size of a block blob	50,000 X 100 MiB (approximately 4.75 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
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
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.

## 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

1/8/2020 • 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.

## Scale targets for standard storage accounts

The following table describes default limits for Azure general-purpose v1, v2, Blob storage, block blob storage, and Data Lake Storage Gen2 enabled 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.

RESOURCE	LIMIT
Number of storage accounts per region per subscription, including standard, premium, and Data Lake Storage Gen2 enabled storage accounts. <sup>3</sup>	250
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
Maximum request rate <sup>1</sup> per storage account	20,000 requests per second
Maximum ingress <sup>1</sup> per storage account (US, Europe regions)	25 Gbps
Maximum ingress <sup>1</sup> per storage account (regions other than US and Europe)	5 Gbps if RA-GRS/GRS is enabled, 10 Gbps for LRS/ZRS <sup>2</sup>
Maximum egress for general-purpose v2 and Blob storage accounts (all regions)	50 Gbps
Maximum egress for general-purpose v1 storage accounts (US regions)	20 Gbps if RA-GRS/GRS is enabled, 30 Gbps for LRS/ZRS <sup>2</sup>
Maximum egress for general-purpose v1 storage accounts (non-US regions)	10 Gbps if RA-GRS/GRS is enabled, 15 Gbps for LRS/ZRS <sup>2</sup>

RESOURCE	LIMIT
Maximum number of virtual network rules per storage account	200
Maximum number of IP address rules per storage account	200

<sup>1</sup> Azure Storage standard accounts support higher capacity limits and higher limits for ingress by request. To request an increase in account limits, contact [Azure Support](#).

<sup>2</sup> If your storage account has read-access enabled with geo-redundant storage (RA-GRS) or geo-zone-redundant storage (RA-GZRS), then the egress targets for the secondary location are identical to those of the primary location. [Azure Storage replication](#) options include:

- [Locally redundant storage \(LRS\)](#)
- [Zone-redundant storage \(ZRS\)](#)
- [Geo-redundant storage \(GRS\)](#)
- [Read-access geo-redundant storage \(RA-GRS\)](#)
- [Geo-zone-redundant storage \(GZRS\)](#)
- [Read-access geo-zone-redundant storage \(RA-GZRS\)](#)

<sup>3</sup> [Azure Data Lake Storage Gen2](#) is a set of capabilities dedicated to big data analytics, built on Azure Blob storage. Azure Storage and blob storage limitations apply to Data Lake Storage Gen2.

#### NOTE

Microsoft recommends that you use a general-purpose v2 storage account for most scenarios. You can easily upgrade a general-purpose v1 or an Azure Blob storage account to a general-purpose v2 account with no downtime and without the need to copy data. For more information, see [Upgrade to a general-purpose v2 storage account](#).

If the needs of your application exceed the scalability targets of a single storage account, you can build your application to use multiple storage accounts. You can then partition your data objects across those storage accounts. For information on volume pricing, see [Azure Storage pricing](#).

All storage accounts run on a flat network topology regardless of when they were created. For more information on the Azure Storage flat network architecture and on scalability, see [Microsoft Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency](#).

## See also

- [Scalability targets for the Azure Storage resource provider](#)
- [Azure subscription limits and quotas](#)

# Scalability targets for premium block blob storage accounts

12/23/2019 • 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](#).

# Scalability and performance targets for premium page blob storage accounts

1/24/2020 • 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.

## 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 Windows 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	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

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 Windows VMs](#) or [Azure Disk Storage overview for Linux 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

1/8/2020 • 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.

## 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](#)

# azcopy

11/13/2019 • 2 minutes to read • [Edit Online](#)

AzCopy is a command-line tool that moves data into and out of Azure Storage.

## 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](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Options

**--cap-mbps uint32** 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** Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

## 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 logout](#)
- [azcopy make](#)
- [azcopy remove](#)
- [azcopy sync](#)

# azcopy bench

10/16/2019 • 2 minutes to read • [Edit Online](#)

Runs a performance benchmark by uploading test data to a specified destination. The test data is automatically generated.

The benchmark command runs the same upload process as 'copy', except that:

- There's no source parameter. The command requires only a destination URL. In the current release, this destination URL must refer to a blob container.
- 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 is not used.
- Only a few of the optional parameters that are available to the copy command are supported.
- Additional diagnostics are measured and reported.
- By default, the transferred data is deleted at the end of the test run.

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 AZCOPY\_CONCURRENCY\_VALUE environment variable to a specific number of connections.

All the usual authentication types are supported. However, the most convenient approach for benchmarking is typically to create an empty container with a SAS token and use SAS authentication.

## Examples

```
azcopy bench [destination] [flags]
```

Run a benchmark test with default parameters (suitable for benchmarking networks up to 1 Gbps):'

- azcopy bench "https://[account].blob.core.windows.net/[container]?"

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]?" --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 "https://[account].blob.core.windows.net/[container]?" --file-count 50000 --size-per-file 8M --put-md5

## 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 - e.g. 0.25. Identical to the same-named parameter in the copy command.

**--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 - e.g. to use it for manual tests outside benchmark mode (default true).

**--file-count** uint The 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")

**--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** uint32 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").

## See also

- [azcopy](#)

# azcopy copy

4/10/2020 • 10 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 <-> ADLS Gen 2 (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)

Please refer to the examples for more information.

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## 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 is 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. Please 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 have not 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]"

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

Download a single file by using OAuth authentication. If you have not 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]" > "/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

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. You have to use a SAS token at the end of the source account URL, but the destination account doesn't need one if you log into AzCopy by using the azcopy login command.

- azcopy cp "https://[srcaccount].blob.core.windows.net/[container]/[path/to/blob]?[SAS]" "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

Please 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

## Options

**--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 (e.g. has administrator rights or is a member of the 'Backup Operators' group). All this flag does is activate privileges that the account already has.

**--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 blobs directly to the [access tier](#) of your choice. (default 'None'). Valid values include 'None', 'Hot', 'Cool', and 'Archive'. If 'None' or no tier is passed, the blob will inherit the tier of the storage account.

**--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 is 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")

**--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.

**--decompress** Automatically decompress files when downloading, if their content-encoding indicates that they are 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.

**--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 is not 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 does not support wildcard characters (\*).

Checks relative path prefix(For example: myFolder;myFolder/subDirName/file.pdf). When used in combination with account traversal, paths do not include the container name.

**--exclude-pattern** string Exclude these files when copying. This option supports wildcard characters (\*)

**--follow-symlinks** Follow symbolic links when uploading from local file system.

**--from-to** string Optionally specifies the source destination combination. For Example: LocalBlob, BlobLocal, LocalBlobFS.

**-h, --help** help for copy

**--include-attributes** string (Windows only) Include files whose attributes match the attribute list. For example: A;S;R

**--include-path** string Include only these paths when copying. This option does not 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 ';'.

**--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")

**--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. Possible values include 'true', 'false', 'ifSourceNewer', and 'prompt'. (default "true")

**--page-blob-tier** string Upload page blob to Azure Storage using this blob tier. (default "None")

**--preserve-last-modified-time** Only available when destination is file system.

**--preserve-smb-permissions** string False by default. Preserves SMB ACLs between aware resources (Windows and Azure Files). For downloads, you will also need to use the **--backup** flag to restore permissions where the new Owner will not be the user that is running AzCopy. This flag applies to both files and folders, unless a file-only filter is specified (e.g. **include-pattern** ).

**--preserve-smb-info** string False 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 (e.g. **include-pattern**). The information transferred for folders is the same as that for files, except for Last Write Time which is never preserved for folders.

**--preserve-owner** Only has an effect in when downloading data, and only when the **--preserve-smb-permissions** is used. If true (the default), the file Owner and Group are preserved in downloads. If this flag is set to false, **--preserve-smb-permissions** will still preserve ACLs but Owner and Group will be based on the user that is running AzCopy.

**--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 sub-directories recursively when uploading from local file system.

**--s2s-detect-source-changed** Check if source has changed after enumerating.

**--s2s-handle-invalid-metadata** string Specifies how invalid metadata keys are handled. Available options: ExcludelfInvalid, FaillfInvalid, RenamelfInvalid. (default "ExcludelfInvalid")

**--s2s-preserve-access-tier** Preserve access tier during service to service copy. Please 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 is not supported, please use s2sPreserveAccessTier=false to bypass copying access tier. (default true)

**--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 additional request per object or file. (default true)

## Options inherited from parent commands

**--cap-mbps uint32** 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")

## See also

- [azcopy](#)

# azcopy doc

11/13/2019 • 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]
```

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Options

OPTION	DESCRIPTION
-h, --help	Shows help content for the doc command.

## Options inherited from parent commands

OPTION	DESCRIPTION
--cap-mbps uint32	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".

## See also

- [azcopy](#)

# azcopy env

11/13/2019 • 2 minutes to read • [Edit Online](#)

Shows the environment variables that can configure AzCopy's behavior.

## Synopsis

```
azcopy env [flags]
```

### 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.

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Options

OPTION	DESCRIPTION
-h, --help	Shows help content for the env command.
--show-sensitive	Shows sensitive/secret environment variables.

## Options inherited from parent commands

OPTION	DESCRIPTION
--cap-mbps uint32	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".

## See also

- [azcopy](#)

# azcopy jobs

11/13/2019 • 2 minutes to read • [Edit Online](#)

Sub-commands 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](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Examples

```
azcopy jobs show [jobID]
```

## Options

OPTION	DESCRIPTION
-h, --help	Show help content for the jobs command.

## Options inherited from parent commands

OPTION	DESCRIPTION
--cap-mbps uint32	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".

## See also

- [azcopy](#)
- [azcopy jobs list](#)
- [azcopy jobs resume](#)
- [azcopy jobs show](#)

# azcopy jobs clean

11/13/2019 • 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](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## 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: Canceled, Completed, Failed, InProgress, All (default "All")

## Options inherited from parent commands

**--cap-mbps** uint32 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")

## See also

- [azcopy jobs](#)

# azcopy jobs list

11/13/2019 • 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](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Options

OPTION	DESCRIPTION
-h, --help	Show help content for the list command.

## Options inherited from parent commands

OPTION	DESCRIPTION
--cap-mbps uint32	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".

## See also

- [azcopy jobs](#)

# azcopy jobs remove

11/13/2019 • 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](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Examples

```
azcopy jobs rm e52247de-0323-b14d-4cc8-76e0be2e2d44
```

## Options

**-h, --help** Help for remove.

## Options inherited from parent commands

**--cap-mbps uint32** 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")

## See also

- [azcopy jobs](#)

# azcopy jobs resume

11/13/2019 • 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](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Options

OPTION	DESCRIPTION
--destination-sas string	Destination SAS of the destination for given JobId.
--exclude string	Filter: Exclude these failed transfer(s) when resuming the job. Files should be separated by ':'.
-h, --help	Show help content for the resume command.
--include string	Filter: only include these failed transfer(s) when resuming the job. Files should be separated by ':'.
--source-sas string	source SAS of the source for given JobId.

## Options inherited from parent commands

OPTION	DESCRIPTION
--cap-mbps uint32	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".

## See also

- [azcopy jobs](#)

# azcopy jobs show

11/13/2019 • 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](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Options

OPTION	DESCRIPTION
<code>-h, --help</code>	Shows help content for the show command.
<code>--with-status string</code>	Only list the transfers of job with this status, available values: Started, Success, Failed

## Options inherited from parent commands

OPTION	DESCRIPTION
<code>--cap-mbps uint32</code>	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.
<code>--output-type string</code>	Format of the command's output. The choices include: text, json. The default value is "text".

## See also

- [azcopy jobs](#)

# azcopy list

11/13/2019 • 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](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Examples

```
azcopy list [containerURL]
```

## Options

OPTION	DESCRIPTION
-h, --help	Show help content for the list command.
--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.

## Options inherited from parent commands

OPTION	DESCRIPTION
--cap-mbps uint32	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".

## See also

- [azcopy](#)

# azcopy login

3/26/2020 • 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.

Please refer to the examples for more information.

### 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](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## 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 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
```

Log in as a service principal using a certificate and password. Set the environment variable AZCOPY\_SPA\_CERT\_PASSWORD to the certificate's password for cert-based service principal authorization.

```
azcopy login --service-principal --certificate-path /path/to/my/cert
```

Make sure to treat /path/to/my/cert as a path to a PEM or PKCS12 file. AzCopy does not reach into the system cert store to obtain your certificate.

--certificate-path is mandatory when doing cert-based service principal auth.

## Options

OPTION	DESCRIPTION
--aad-endpoint	The Azure Active Directory endpoint to use. The default ( <a href="https://login.microsoftonline.com">https://login.microsoftonline.com</a> ) is correct for the public Azure cloud. Set this parameter when authenticating in a national cloud. See <a href="#">Azure AD authentication endpoints</a> .
This flag is 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	Show help content for the login command.
--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.

OPTION	DESCRIPTION
--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 SPN (Service Principal Name) by using a certificate or a secret. The client secret or certificate password must be placed in the appropriate environment variable. Type <code>AzCopy env</code> 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

OPTION	DESCRIPTION
--cap-mbps uint32	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".

## See also

- [azcopy](#)

# azcopy logout

11/13/2019 • 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](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Options

OPTION	DESCRIPTION
-h, --help	Show help content for the logout command.

## Options inherited from parent commands

OPTION	DESCRIPTION
--cap-mbps uint32	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".

## See also

- [azcopy](#)

# azcopy make

11/13/2019 • 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](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Examples

```
azcopy make "https://[account-name].[blob,file,dfs].core.windows.net/[top-level-resource-name]"
```

## Options

OPTION	DESCRIPTION
-h, --help	Show help content for the make command.
--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

OPTION	DESCRIPTION
--cap-mbps uint32	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".

## See also

- [azcopy](#)

# azcopy remove

11/13/2019 • 2 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](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Examples

Remove a single blob with SAS:

```
azcopy rm "https://[account].blob.core.windows.net/[container]/[path/to/blob]?[SAS]"
```

Remove an entire virtual directory with a SAS:

```
azcopy rm "https://[account].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" --recursive=true
```

Remove only the top blobs inside a virtual directory but not its sub-directories:

```
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: 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="*.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="foo*,*bar"
```

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
file content:
dir1/dir2
blob1
blob2
```

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 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

**--exclude-path** string Exclude these paths when removing. This option does not 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

**-h, --help** help for remove

**--include-path** string Include only these paths when removing. This option does not 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.

**--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")

**--recursive** Look into sub-directories recursively when syncing between directories.

## Options inherited from parent commands

OPTION	DESCRIPTION
--cap-mbps uint32	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".

## See also

- [azcopy](#)

# azcopy sync

4/22/2020 • 4 minutes to read • [Edit Online](#)

Replicates the source location to the destination location.

## 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 (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)

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 'deleteDestination' flag is set to true or prompt, then sync will delete files and blobs at the destination that are not present at the source.

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Advanced

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 (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.

```
azcopy sync <source> <destination> [flags]
```

# Examples

Sync a single file:

```
azcopy sync "/path/to/file.txt" "https://[account].blob.core.windows.net/[container]/[path/to/blob]"
```

## NOTE

The destination blob *must* exist. Use `azcopy copy` to copy a single file that does not yet exist in the destination. Otherwise, the following error occurs:

```
Cannot perform sync due to error: sync must happen between source and destination of the same type, e.g.
either file <-> file, or directory/container <-> directory/container
```

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 sync "/path/to/file.txt" "https://[account].blob.core.windows.net/[container]/[path/to/blob]" --put-md5
```

Sync an entire directory including its sub-directories (note that recursive is on by default):

```
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 top files inside a directory but not its sub-directories:

```
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="*.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="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/exclude flags are used together, only files matching the include patterns would be looked at, but those matching the exclude patterns would be always be 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")

**--delete-destination** string Defines whether to delete extra files from the destination that are not 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")

**--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 copying. This option does not support wildcard characters (\*). Checks relative path prefix(For example: myFolder;myFolder/subDirName/file.pdf). When used in combination with account traversal, paths do not include the container name.

**--exclude-pattern** string Exclude files where the name matches the pattern list. For example: \*.jpg;\*.pdf;exactName  
**-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

**--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")

**--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 sub-directories recursively when syncing between directories. (default true).  
(default true)

## Options inherited from parent commands

OPTION	DESCRIPTION
--cap-mbps uint32	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".

## See also

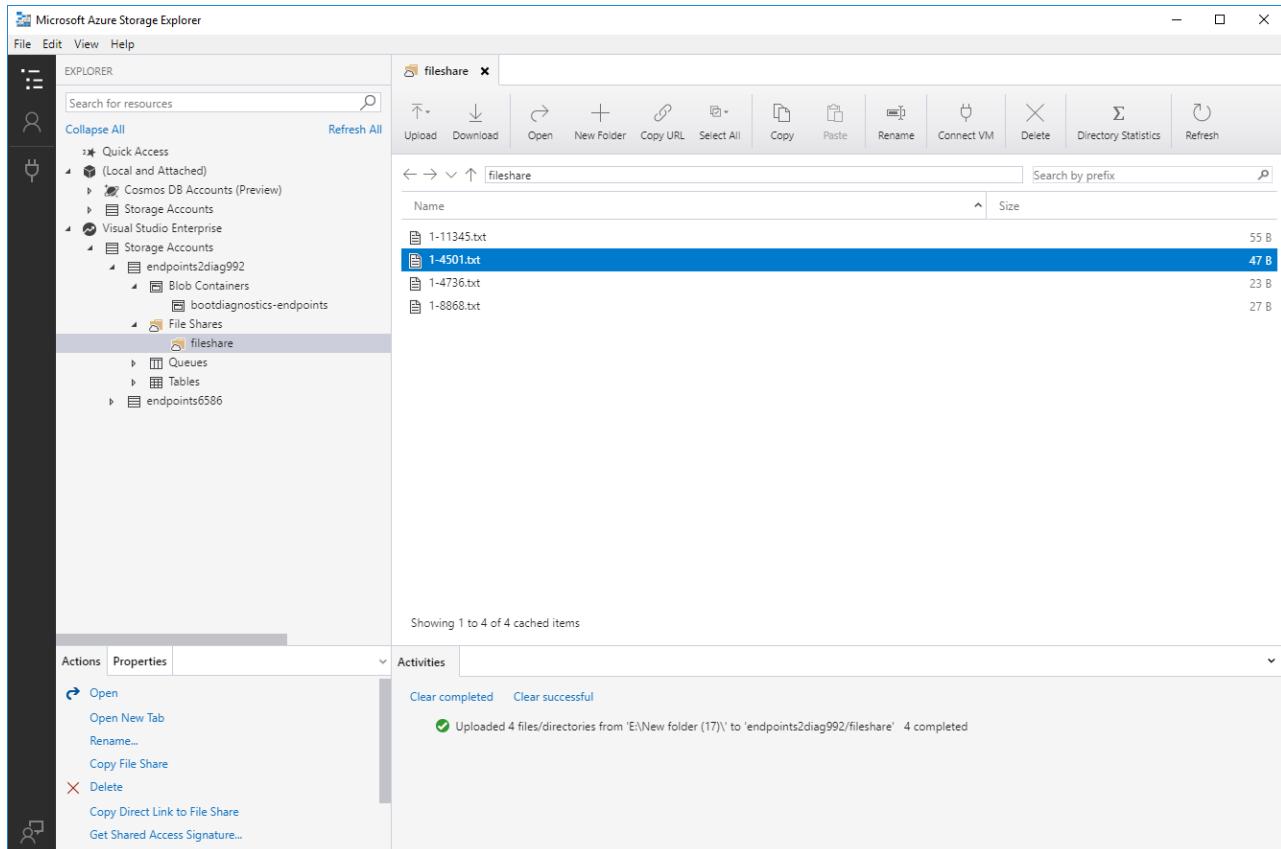
- [azcopy](#)

# Get started with Storage Explorer

11/8/2019 • 10 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 10 (recommended)
- Windows 8
- Windows 7

For all versions of Windows, Storage Explorer requires .NET Framework 4.6.2 or later.

## 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 storage accounts. In general you can either:

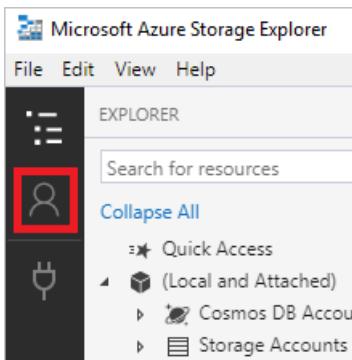
- [Sign in to Azure to access your subscriptions and their resources](#)
- [Attach a specific Storage or CosmosDB resource](#)

## Sign in to Azure

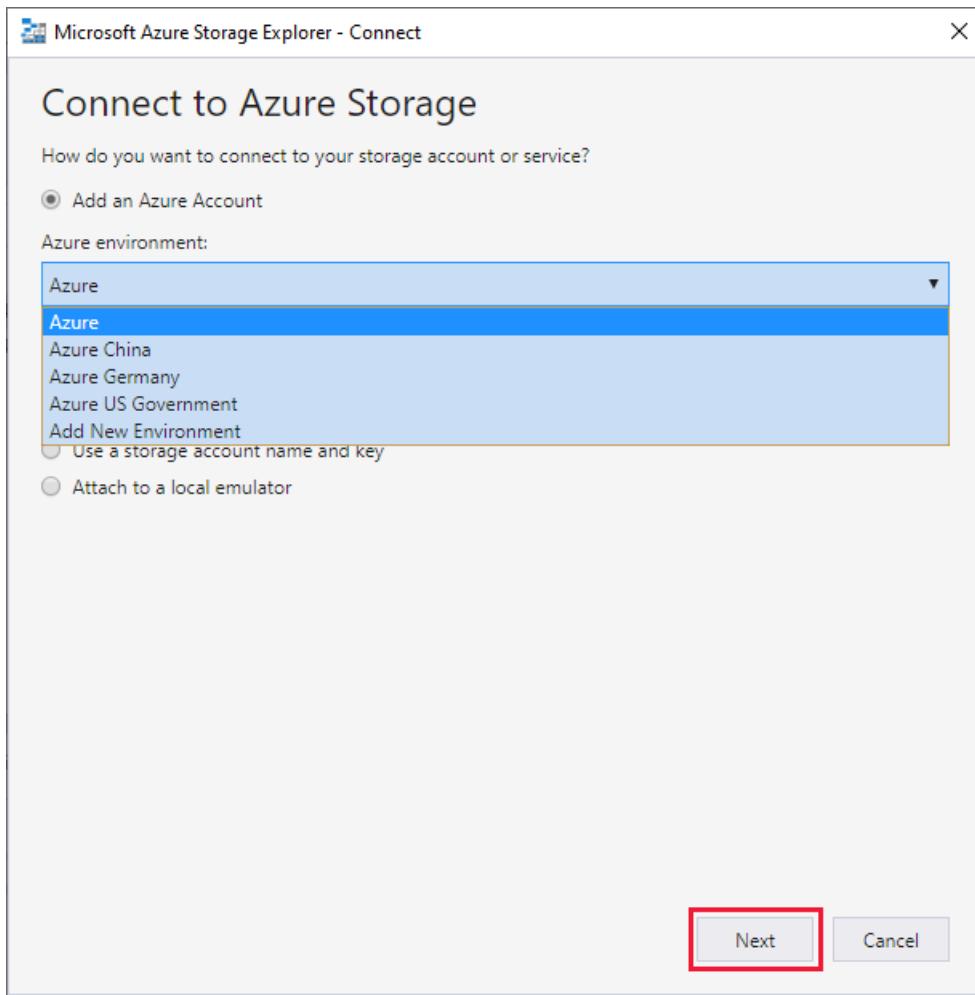
### NOTE

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, which give you access to your storage account, the containers in the account, and the data in the containers. If you have permissions only at the data layer, consider [adding a resource through Azure AD](#). 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.

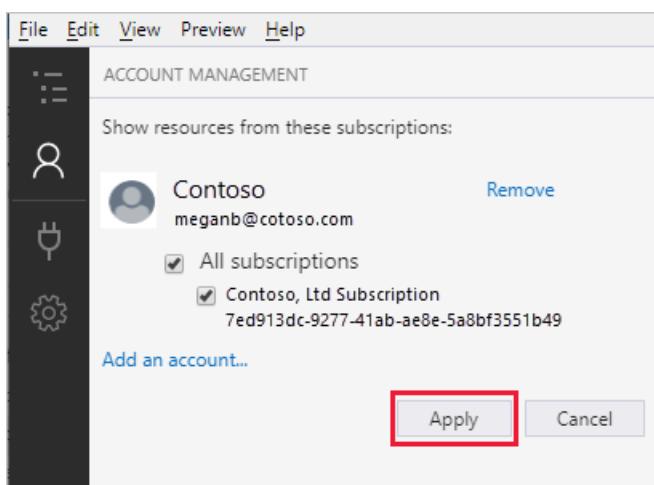


2. **ACCOUNT MANAGEMENT** now displays all the Azure accounts you've signed in to. To connect to another account, select **Add an account**.
3. In **Connect to Azure Storage**, select an Azure cloud from **Azure environment** to sign in to a national cloud or an Azure Stack. After you choose your environment, select **Next**.

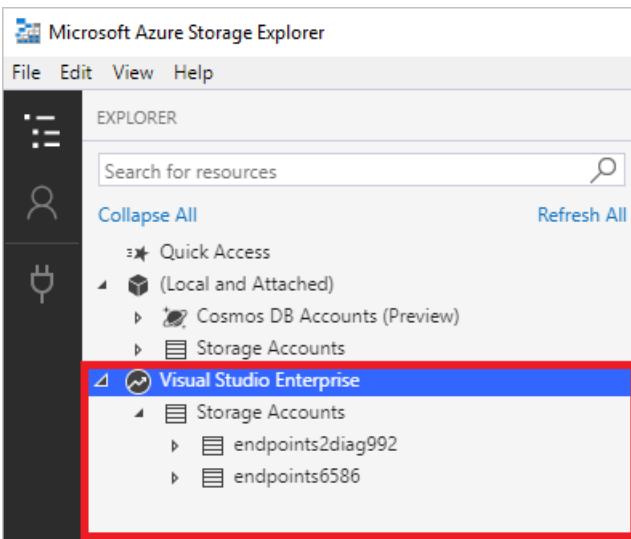


Storage Explorer opens a page for you to sign in. For more information, see [Connect storage explorer to an Azure Stack subscription or storage account](#).

- After you successfully sign in with an Azure account, the account and the Azure subscriptions associated with that account appear under **ACCOUNT MANAGEMENT**. Select **All subscriptions** to toggle your selection between all or none of the listed Azure subscriptions. Select the Azure subscriptions that you want to work with, and then select **Apply**.



EXPLORER displays the storage accounts associated with the selected Azure subscriptions.



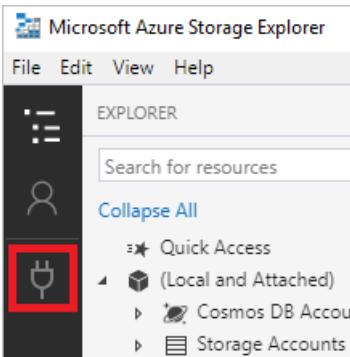
## Attach a specific resource

There are several ways to attach to a resource in Storage Explorer:

- [Add a resource via Azure AD](#). If you have permissions only at the data layer, use this option to add a blob container or an Azure Data Lake Storage Gen2 Blob storage container.
- [Use a connection string](#). Use this option if you have a connection string to a storage account. Storage Explorer supports both key and [shared access signature](#) connection strings.
- [Use a shared access signature URI](#). If you have a [shared access signature URI](#) to a blob container, file share, queue, or table, use it to attach to the resource. To get a shared access signature URI, you can either use [Storage Explorer](#) or the [Azure portal](#).
- [Use a name and key](#). If you know either of the account keys to your storage account, you can use this option to quickly connect. Find your keys in the storage account page by selecting **Settings > Access keys** in the [Azure portal](#).
- [Attach to a local emulator](#). If you're using one of the available Azure Storage emulators, use this option to easily connect to your emulator.
- [Connect to an Azure Cosmos DB account by using a connection string](#). Use this option if you have a connection string to a CosmosDB instance.
- [Connect to Azure Data Lake Store by URI](#). Use this option if you have a URI to Azure Data Lake Store.

### Add a resource via Azure AD

1. Select the Connect symbol to open **Connect to Azure Storage**.



2. If you haven't already done so, use the **Add an Azure Account** option to sign in to the Azure account that has access to the resource. After you sign in, return to **Connect to Azure Storage**.
3. Select **Add a resource via Azure Active Directory (Azure AD)**, and then select **Next**.
4. Select an Azure account and tenant. These values must have access to the Storage resource you want to attach to. Select **Next**.

5. Choose the resource type you want to attach. Enter the information needed to connect.

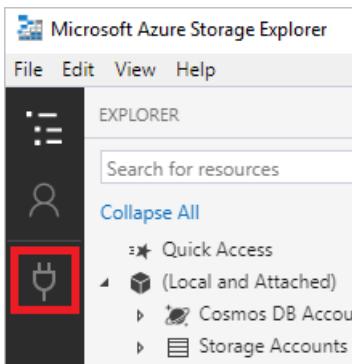
The information you enter on this page depends on what type of resource you're adding. Make sure to choose the correct type of resource. After you've entered the required information, select **Next**.

6. Review the **Connection Summary** to make sure all the information is correct. If it is, select **Connect**. Otherwise, select **Back** to return to the previous pages to fix any incorrect information.

After the connection is successfully added, the resource tree goes to the node that represents the connection. The resource appears under **Local & Attached > Storage Accounts > (Attached Containers) > 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](#).

#### Use a connection string

1. Select the **Connect** symbol to open **Connect to Azure Storage**.



2. Select **Use a connection string**, and then select **Next**.

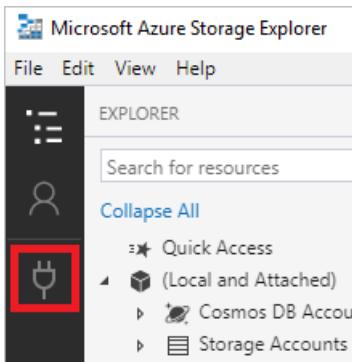
3. Choose a display name for your connection and enter your connection string. Then, select **Next**.

4. Review the **Connection Summary** to make sure all the information is correct. If it is, select **Connect**. Otherwise, select **Back** to return to the previous pages to fix any incorrect information.

After the connection is successfully added, the resource tree goes to the node that represents the connection. The resource appears under **Local & Attached > Storage Accounts**. 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](#).

#### Use a shared access signature URI

1. Select the **Connect** symbol to open **Connect to Azure Storage**.



2. Select **Use a shared access signature (SAS) URI**, and then select **Next**.

3. Choose a display name for your connection and enter your shared access signature URI. The service endpoint for the type of resource you're attaching should autofill. If you're using a custom endpoint, it's possible it might not. Select **Next**.

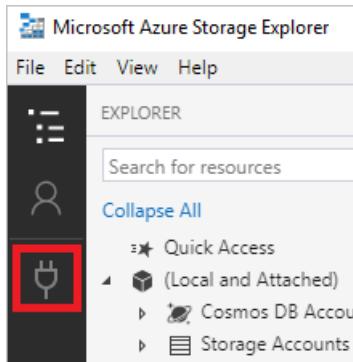
4. Review the **Connection Summary** to make sure all the information is correct. If it is, select **Connect**.

Otherwise, select **Back** to return to the previous pages to fix any incorrect information.

After the connection is successfully added, the resource tree goes to the node that represents the connection. The resource appears under **Local & Attached > Storage Accounts > (Attached Containers) > the service node for the type of container you attached**. If Storage Explorer couldn't add your connection, see the [Azure Storage Explorer troubleshooting guide](#). See the troubleshooting guide if you can't access your data after successfully adding the connection.

#### Use a name and key

1. Select the **Connect** symbol to open **Connect to Azure Storage**.



2. Select **Use a storage account name and key**, and then select **Next**.
3. Choose a display name for your connection.
4. Enter your storage account name and either of its access keys.
5. Choose the **Storage domain** to use and then select **Next**.
6. Review the **Connection Summary** to make sure all the information is correct. If it is, select **Connect**. Otherwise, select **Back** to return to the previous pages to fix any incorrect information.

After the connection is successfully added, the resource tree goes to the node that represents the connection. The resource appears under **Local & Attached > Storage Accounts**. 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](#).

#### Attach to a local emulator

Storage Explorer currently supports two official Storage emulators:

- [Azure Storage emulator](#) (Windows only)
- [Azurite](#) (Windows, macOS, or Linux)

If your emulator is listening on the default ports, you can use the **Emulator - Default Ports** node to access your emulator. Look for **Emulator - Default Ports** under **Local & Attached > Storage Accounts**.

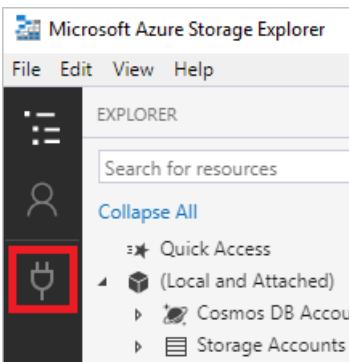
If you want to use a different name for your connection, or if your emulator isn't running on the default ports, follow these steps:

1. Start your emulator. Enter the command `AzureStorageEmulator.exe status` to display the ports for each service type.

#### IMPORTANT

Storage Explorer doesn't automatically start your emulator. You must start it manually.

2. Select the **Connect** symbol to open **Connect to Azure Storage**.



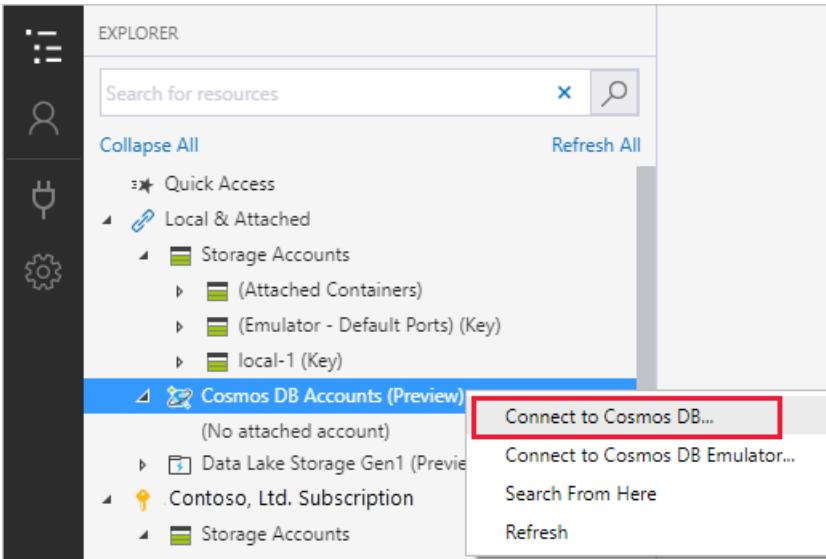
3. Select **Attach to a local emulator**, and then select **Next**.
4. Choose a display name for your connection and enter the ports your emulator is listening on for each service type. **Attach to a Local Emulator** suggests the default port values for most emulators. **Files port** is blank, because neither of the official emulators currently support the Files service. If the emulator you're using does support Files, you can enter the port to use. Then, select **Next**.
5. Review the **Connection Summary** and make sure all the information is correct. If it is, select **Connect**. Otherwise, select **Back** to return to the previous pages to fix any incorrect information.

After the connection is successfully added, the resource tree goes to the node that represents the connection. The node should appear under **Local & Attached > Storage Accounts**. 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](#).

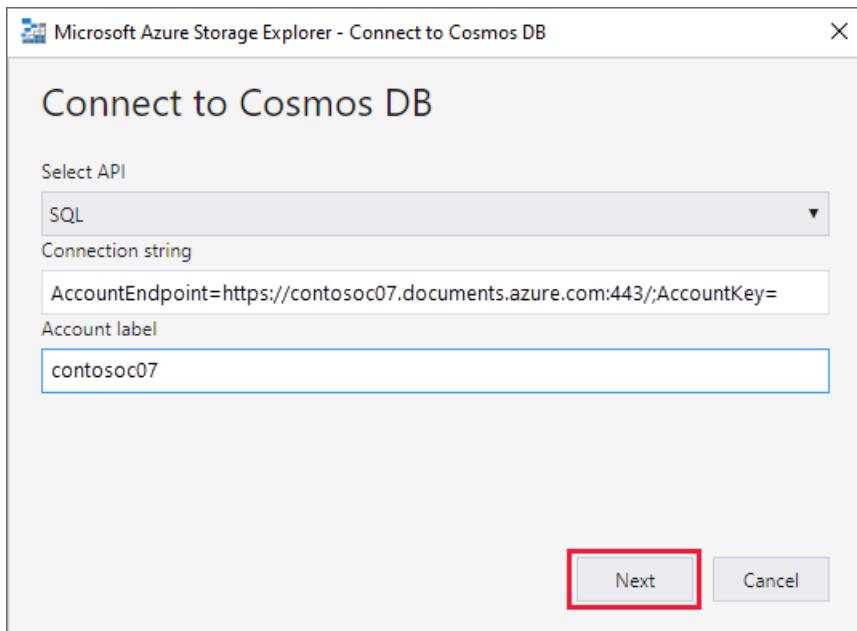
#### **Connect to an Azure Cosmos DB account by using a connection string**

Instead of managing Azure Cosmos DB accounts through an Azure subscription, you can connect to Azure Cosmos DB by using a connection string. To connect, follow these steps:

1. Under EXPLORER, expand **Local & Attached**, right-click **Cosmos DB Accounts**, and select **Connect to Azure Cosmos DB**.



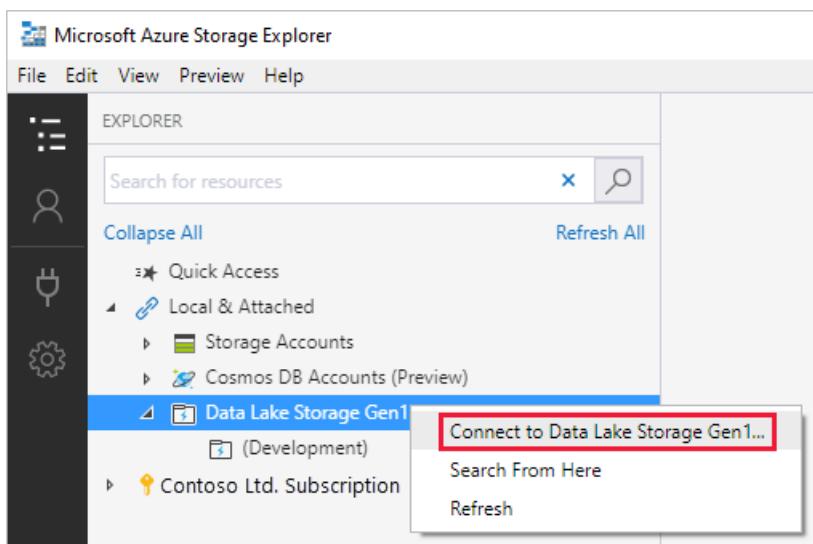
2. Select the Azure Cosmos DB API, enter your **Connection String** data, and then select **OK** to connect the Azure Cosmos DB account. For information about how to retrieve the connection string, see [Manage an Azure Cosmos account](#).



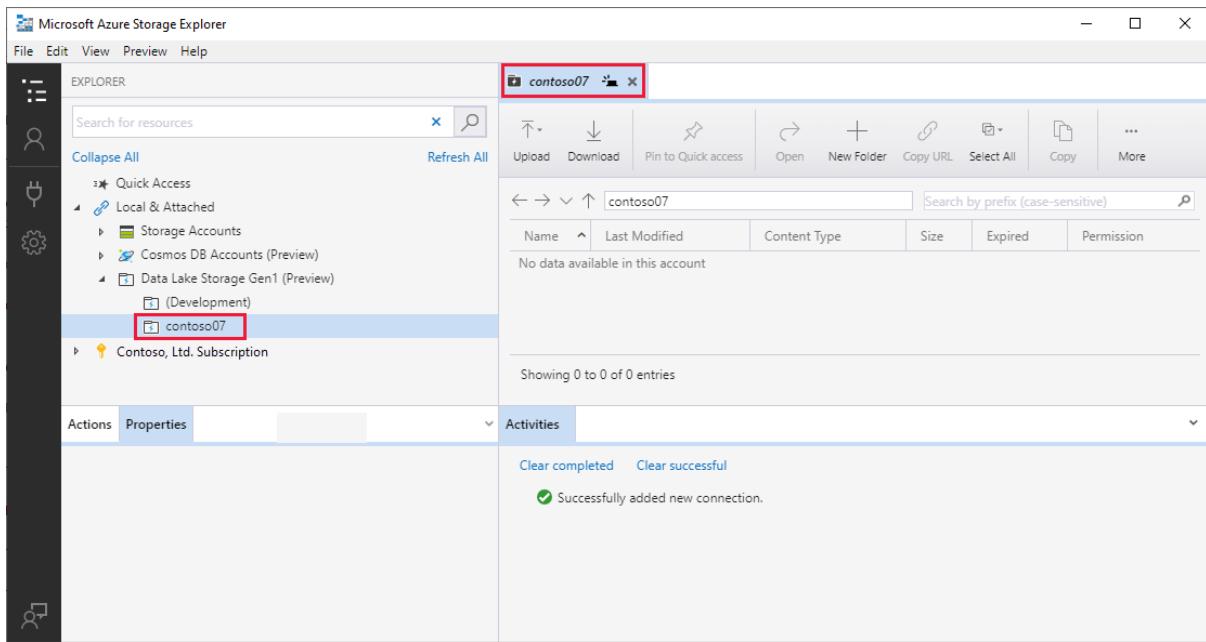
#### Connect to Azure Data Lake Store by URI

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.

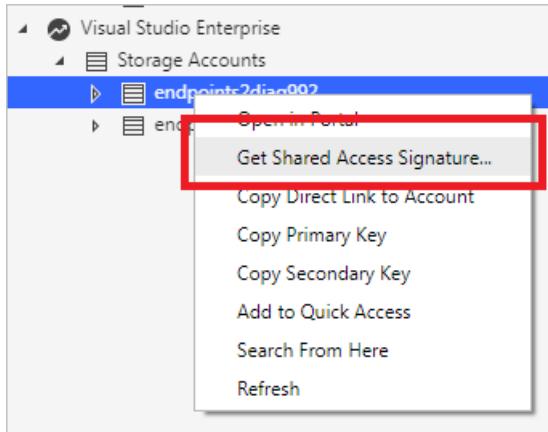


This example uses Data Lake Storage Gen1. Azure Data Lake Storage Gen2 is now available. For more information, see [What is Azure Data Lake Storage Gen1](#).

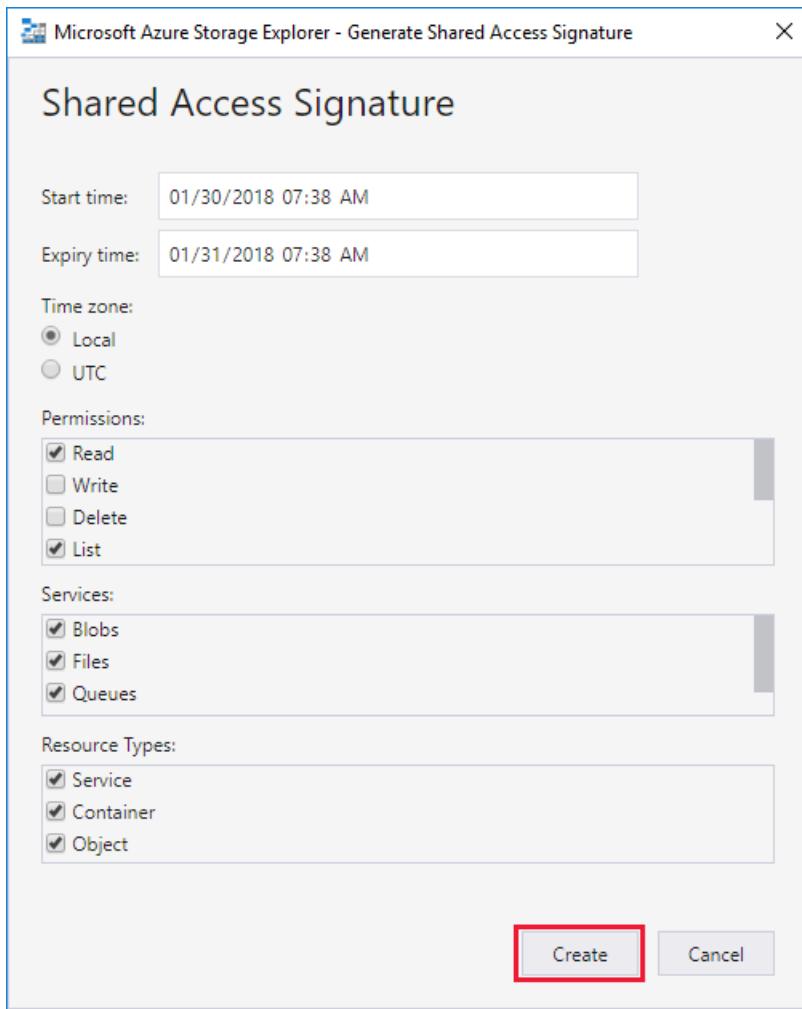
## Generate a shared access signature in Storage Explorer

### 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.

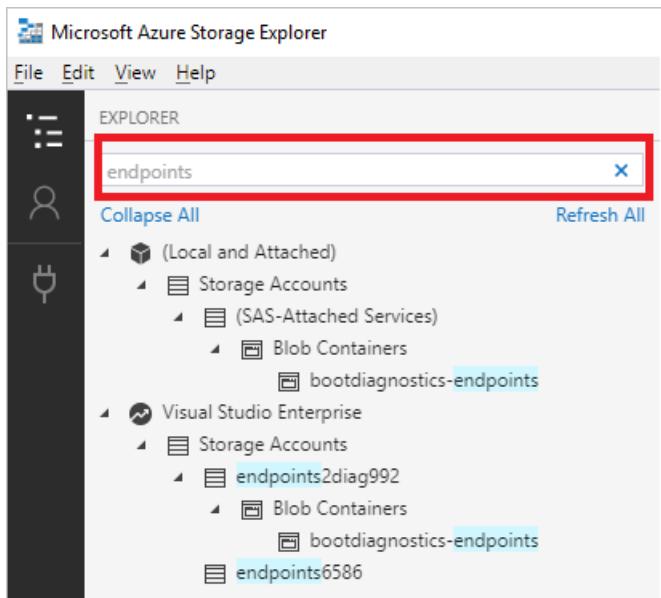
#### Service level shared access signature

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](#)
- [Work with data using Azure Storage Explorer](#)
- [Manage Azure Data Lake Store resources with Storage Explorer](#)

# Azure Storage Explorer troubleshooting guide

3/31/2020 • 17 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.

## RBAC permissions issues

Role-based access control [RBAC](#) enables highly granular access management of Azure resources by combining sets of permissions into *roles*. Here are some strategies to get RBAC working optimally in Storage Explorer.

### How do I access my resources in Storage Explorer?

If you're having problems accessing storage resources through 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 administrator 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. Therefore, 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.

RBAC roles can contain 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 administrator?

We don't currently have an RBAC-related solution for this issue. As a workaround, you can request a SAS URI to attach to your resource.

### Recommended built-in RBAC roles

There are several built-in RBAC roles which can provide the permissions needed to use Storage Explorer. Some of those roles are:

- [Owner](#): Manage everything, including access to resources. **Note**: this role will give you key access.
- [Contributor](#): Manage everything, excluding access to resources. **Note**: this role will give you key access.
- [Reader](#): Read and list resources.
- [Storage Account Contributor](#): Full management of storage accounts. **Note**: this role will give you key access.
- [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.

## Error: Self-signed certificate in certificate chain (and similar errors)

Certificate errors typically occur in one of the following situations:

- The app is connected through a *transparent proxy*, which means a server (such as your company server) is intercepting HTTPS traffic, decrypting it, and then encrypting it by using a self-signed certificate.
- You're running an application that's injecting a self-signed TLS/SSL certificate into the HTTPS messages that you receive. Examples of applications that inject certificates include antivirus and network traffic inspection software.

When Storage Explorer sees a self-signed or untrusted certificate, it no longer knows whether the received HTTPS message has been altered. If you have a copy of the self-signed certificate, you can instruct Storage Explorer to trust it by following these steps:

1. Obtain a Base-64 encoded X.509 (.cer) copy of the certificate.
2. Go to **Edit > SSL Certificates > Import Certificates**, and then use the file picker to find, select, and open the .cer file.

This issue may also occur if there are multiple certificates (root and intermediate). To fix this error, both certificates must be added.

If you're unsure of where the certificate is coming from, follow these steps to find it:

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 `s_client -showcerts -connect microsoft.com:443`.
4. Look for self-signed certificates. If you're unsure of which certificates are self-signed, make note of anywhere the subject `("s:")` and issuer `("i:")` are the same.
5. When you find self-signed certificates, for each one, copy and paste everything from (and including) `-----BEGIN CERTIFICATE-----` through `-----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 that you created.

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 by using the `--ignore-certificate-errors` flag. When opened with this flag, Storage Explorer ignores certificate errors.

## Sign-in issues

### Blank sign-in dialog box

Blank sign-in dialog boxes most often occur when Active Directory Federation Services (AD FS) prompts Storage Explorer to perform a redirect, which is unsupported by Electron. To work around this issue, you can try to use Device Code Flow for sign-in. To do so, follow these steps:

1. On the left vertical tool bar, open **Settings**. In the Settings Panel, go to **Application > Sign in**. Enable **Use device code flow sign-in**.
2. Open the **Connect** dialog box (either through the plug icon on the left-side vertical bar or by selecting **Add Account** on the account panel).
3. Choose the environment you want to sign in to.
4. Select **Sign In**.
5. Follow the instructions on the next panel.

If you can't sign in to the account you want to use because your default browser is already signed in to a different account, do one of the following:

- Manually copy the link and code into a private session of your browser.
- Manually copy the link and code into a different browser.

### Reauthentication loop or UPN change

If you're in a reauthentication loop or have changed the UPN of one of your accounts, follow these steps:

1. Remove all accounts and then close Storage Explorer.
2. 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.
3. If you're running Mac or Linux, you'll 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.

### Conditional Access

Because of a limitation in the Azure AD Library used by Storage Explorer, Conditional Access isn't supported when Storage Explorer is being used on Windows 10, Linux, or macOS.

## Mac Keychain errors

The macOS Keychain can sometimes enter a state that causes issues for the Storage Explorer authentication library. To get the Keychain out of this state, follow these steps:

1. Close Storage Explorer.
2. Open Keychain (press Command+Spacebar, type `keychain`, and press Enter).
3. Select the "login" Keychain.
4. Select the padlock icon to lock the Keychain. (The padlock will appear locked when the process is complete. 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 **Allow** if **Always Allow** isn't available).
7. Try to sign in.

#### General sign-in troubleshooting steps

- If you're on macOS, and the sign-in window never appears over the **Waiting for authentication** dialog box, try [these steps](#).
- Restart Storage Explorer.
- If the authentication window is blank, wait at least one minute before closing the authentication dialog box.
- Make sure that your proxy and certificate settings are properly configured for both your machine and Storage Explorer.
- If you're running Windows and have access to Visual Studio 2019 on the same machine and to the sign-in credentials, try signing in to Visual Studio 2019. After a successful sign-in to Visual Studio 2019, you can open Storage Explorer and see your account in the account panel.

If none of these methods work, [open an issue in 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 (Azure, Azure China 21Vianet, Azure Germany, Azure US Government, or Custom Environment).
- If you're behind a proxy server, make sure you've configured the Storage Explorer proxy correctly.
- Try removing and re-adding the account.
- If there's a "More information" 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, feel free to [open an issue in GitHub](#).

## Can't remove an attached account or storage 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`

#### NOTE

Close Storage Explorer before you delete these folders.

#### NOTE

If you have 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

First, make sure that the following information you entered is correct:

- The proxy URL and port number
- Username and password if the proxy requires them

#### NOTE

Storage Explorer doesn't support proxy auto-config files for configuring proxy settings.

## Common solutions

If you're still experiencing issues, try the following troubleshooting methods:

- If you can connect to the internet without using your proxy, verify that Storage Explorer works without proxy settings enabled. If this is the case, there may be an issue with your proxy settings. Work with your administrator to identify the problems.
- Verify that other applications that use the proxy server work as expected.
- Verify that you can connect to the portal for the Azure environment you're trying to use.
- 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 `InvalidParameterValue` or a similar XML response.
- If someone else is also using Storage Explorer with your proxy server, verify that they can connect. If they can, you may have to contact your proxy server admin.

## Tools for diagnosing issues

If you have networking tools, such as Fiddler for Windows, you can diagnose the problems as follows:

- If you have to work through your proxy, you may have to configure your networking tool to connect through the proxy.
- Check the port number used by your networking tool.
- Enter the local host URL and the networking tool's port number as proxy settings in Storage Explorer. When you do this correctly, your networking tool starts logging network requests made by Storage Explorer to management and service endpoints. For example, enter `https://cawablobgrs.blob.core.windows.net/` for your blob endpoint in a browser, and you'll receive a response that resembles the following:

```
<?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>
```

This response suggests the resource exists, even though you can't access it.

## Contact proxy server admin

If your proxy settings are correct, you may 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 doesn't currently 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 you're granted access to a resource from the owner of the subscription or account, 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. You can do this by right-clicking the node for your storage account and selecting **Open in Portal**. Then, go to the **Access Keys** blade. 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 SAS 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** (left side) > **file:///**.
3. Depending on the type of connection you're having an issue with, look for its key and then copy its value into a text editor. The value is an array of your custom connection names, like the following:
  - 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 `[]`.

If you want to preserve the connections that aren't corrupted, you can use the following steps to locate the corrupted connections. If you don't mind losing all existing connections, you can 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, and 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 have examined all your connections.

After going through all your connections, for all connections names that aren't added back, you must clear their corrupted data (if there is any) and 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 (for example, `StorageExplorer_CustomConnections_Accounts_v1/account1`).
4. Delete these entries and re-add the connections.

If you still encounter this error after running these steps, or if you want to share what you suspect has corrupted the connections, [open an issue](#) on our GitHub page.

## Issues with SAS URL

If you're connecting to a service through a SAS URL and experiencing an error:

- Verify that the URL provides the necessary permissions to read or list resources.
- Verify that the URL has not expired.
- If the SAS URL is based on an access policy, verify that the access policy has not been revoked.

If you accidentally attached by using an invalid SAS URL and now cannot detach, follow these steps:

1. When you're running Storage Explorer, press F12 to open the Developer Tools window.
2. On the **Application** tab, select **Local Storage** > `file://` in the tree on the left.
3. Find the key associated with the service type of the problematic SAS URI. For example, if the bad SAS 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 then delete it.
5. Press Ctrl+R to reload Storage Explorer.

## Linux dependencies

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, and 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
```

You can also download the application as a .tar.gz file, but you'll have to install dependencies manually.

## IMPORTANT

Storage Explorer as provided in the .tar.gz download is supported only for Ubuntu distributions. Other distributions haven't been verified and may require alternative or additional packages.

These packages are the most common requirements for Storage Explorer on Linux:

- [.NET Core 2.2 Runtime](#)
- `libgconf-2-4`
- `libgnome-keyring0` OR `libgnome-keyring-dev`
- `libgnome-keyring-common`

## NOTE

Storage Explorer version 1.7.0 and earlier require .NET Core 2.0. If you have a newer version of .NET Core installed, you'll have to [patch Storage Explorer](#). If you're running Storage Explorer 1.8.0 or later, you should be able to use up to .NET Core 2.2. Versions beyond 2.2 have not been verified to work at this time.

- [Ubuntu 19.04](#)
- [Ubuntu 18.04](#)
- [Ubuntu 16.04](#)
- [Ubuntu 14.04](#)

1. Download Storage Explorer.
2. Install the [.NET Core Runtime](#).
3. Run the following command:

```
sudo apt-get install libgconf-2-4 libgnome-keyring0
```

## Patching 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](#). 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" from the Azure portal doesn't work

If the **Open In Explorer** button on the Azure portal doesn't work, make sure you're using a compatible browser. The following browsers have been tested for compatibility:

- Microsoft Edge
- Mozilla Firefox
- Google Chrome
- Microsoft Internet Explorer

## Next steps

If none of these solutions work for you, [open an issue in GitHub](#). You can also do this by selecting the **Report issue to GitHub** button in the lower-left corner.

Actions Properties	
URL	<a href="https://">https://</a>
Type	Blob C
Public Read Access	Off
Lease State	availab
Lease Status	unlocke
Last Modified	Mon, 2

# Storage Explorer Accessibility

4/17/2019 • 2 minutes to read • [Edit Online](#)

## Screen Readers

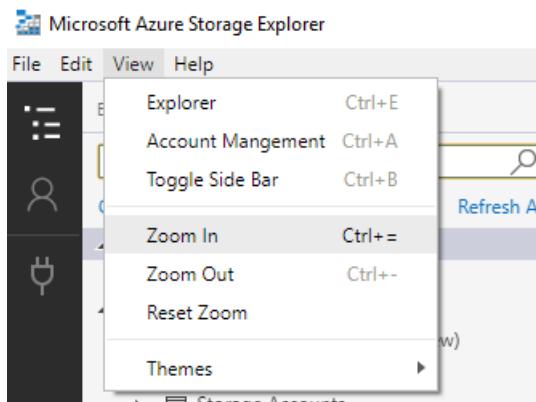
Storage Explorer supports the use of a screen reader on Windows and Mac. The following screen readers are recommended for each platform:

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

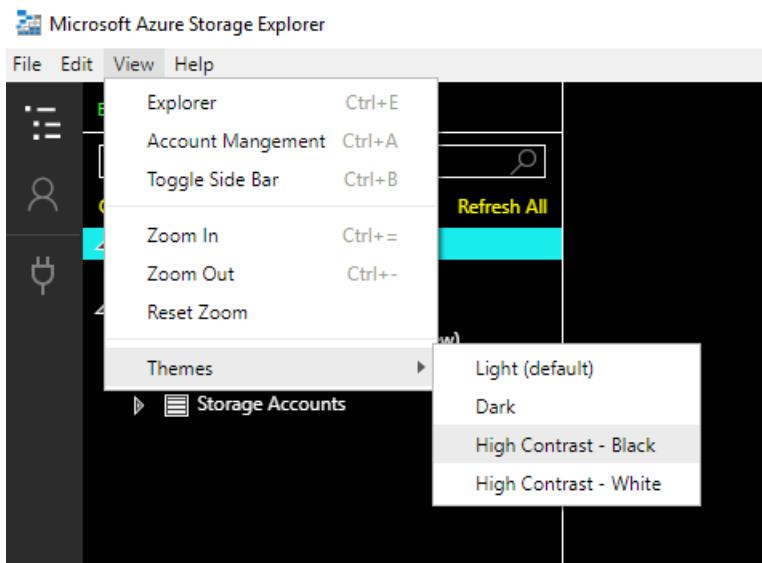
You can make the text in Storage Explorer larger via zooming in. To zoom in, click on **Zoom In** in the Help menu. You can also use the Help menu to zoom out and reset the zoom level back to the default level.



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 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

# Azure Storage compliance offerings

2/1/2019 • 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, 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 \(Netherlands\)](#)
- [NHS IG Toolkit \(UK\)](#)
- [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

3/10/2020 • 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 is the result of converging the capabilities of our two existing storage services, Azure Blob storage and Azure Data Lake Storage Gen1. Features from [Azure Data Lake Storage Gen1](#), such as file system semantics, directory, and file level security and scale are combined with low-cost, tiered storage, high availability/disaster recovery capabilities from [Azure Blob storage](#).

## 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 rather than enumerating and processing 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](#) is available within all Apache Hadoop environments, including [Azure HDInsight](#), [Azure Databricks](#), and [SQL Data Warehouse](#) to access data stored in Data Lake Storage Gen2.
- **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. As data transitions through its complete lifecycle, billing rates change keeping costs to a minimum via built-in features such as [Azure Blob storage 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). Beyond just persistence, processing is executed at near-constant per-request latencies that are measured at the service, account, and file levels.

## Cost effectiveness

One of the many benefits of building Data Lake Storage Gen2 on top of Azure Blob storage is the low cost of storage capacity and transactions. Unlike other cloud storage services, data stored in Data Lake Storage Gen2 is not required to be moved or transformed prior to performing analysis. 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

Data Lake Storage Gen2 is an additional capability for big data analytics, built on top of Azure Blob storage. While there are many benefits in leveraging existing platform components of Blobs to create and operate data lakes for analytics, it does lead to multiple concepts describing 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](#) 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 features.

For a list of supported Blob storage features, see [Blob Storage features available in Azure Data Lake Storage Gen2](#).

## Supported Azure service integrations

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](#).

## 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

- [Known issues with Azure Data Lake Storage Gen2](#)
- [Multi-protocol access on Azure Data Lake Storage](#)

# Quickstart: Analyze data with Databricks

2/19/2020 • 5 minutes to read • [Edit Online](#)

In this quickstart, you run an Apache Spark job using Azure Databricks to perform analytics on data stored in a storage account. As part of the Spark job, you'll analyze a radio channel subscription data to gain insights into free/paid usage based on demographics.

## Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- The name of your Azure Data Lake Gen2 storage account. [Create an Azure Data Lake Storage Gen2 storage account](#).
- The tenant ID, app ID, and password of an Azure service principal with an assigned role of **Storage Blob Data Contributor**. [Create a service principal](#).

### IMPORTANT

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.

## 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**.

The screenshot shows the Microsoft Azure Marketplace interface. On the left, there's a sidebar with various service icons and a 'Create a resource' button. The main area has a search bar at the top and a 'New' tab selected. Below that is a 'Search the Marketplace' input field. The 'Azure Marketplace' tab is active, and under it, the 'Featured' section is shown. A red box highlights the 'Analytics' category, which includes options like HDInsight, Data Lake Analytics, Stream Analytics job, Analysis Services, Azure Databricks, Power BI Embedded, and Integration. Each item has a small icon and a 'Quickstart tutorial' link.

- Under Azure Databricks Service, provide the values to create a Databricks workspace.

This screenshot shows the 'Azure Databricks Service' creation form. It includes fields for 'Workspace name' (set to 'mydatabricksaws'), 'Subscription' (a dropdown menu), 'Resource group' (radio buttons for 'Create new' or 'Use existing', with 'Create new' selected and 'mydatabricksresgrp' entered), 'Location' (set to 'West US 2'), and 'Pricing Tier' (a dropdown menu showing 'Standard (Apache Spark, Secure with Azure AD)' and 'Premium (+ Role-based access controls)'). At the bottom, there's a checked checkbox for 'Pin to dashboard' and two buttons: 'Create' (in blue) and 'Automation options'.

Provide the following values:

PROPERTY	DESCRIPTION
Workspace name	Provide a name for your Databricks workspace
Subscription	From the drop-down, select your Azure subscription.
Resource group	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> .
Location	Select <b>West US 2</b> . Feel free to select another public region if you prefer.
Pricing Tier	Choose between <b>Standard</b> or <b>Premium</b> . For more information on these tiers, see <a href="#">Databricks pricing page</a> .

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 Databricks

1. In the Azure portal, go to the 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. At the top, there's a logo and the text "Azure Databricks". Below it, under "Featured Notebooks", there are three items: "Introduction to Apache Spark on Databricks" (Python icon), "Databricks for Data Scientists" (red server icon), and "Introduction to Structured Streaming" (Python icon). In the center, there are three main sections: "New" (with options: Notebook, Job, Cluster, Table, Library, where "Cluster" is highlighted with a red box), "Documentation" (with links: Databricks Guide, Python, R, Scala, SQL, Importing Data), and "Open Recent" (with a note: "Recent files appear here as you work. Get started with the welcome guide.").

3. In the **New cluster** page, provide the values to create a cluster.

The screenshot shows the 'Create Cluster' interface in the Azure Databricks portal. On the left, there's a sidebar with icons for Home, Workspace, Recents, Data, Clusters, Jobs, and Search. The main area is titled 'New Cluster' with a 'Cancel' button and a prominent blue 'Create Cluster' button. Below these are several configuration fields:

- Cluster Name:** my-spark-cluster
- Cluster Mode:** Standard
- Pool:** None
- Databricks Runtime Version:** Runtime: 5.4 (Scala 2.11, Spark 2.4.3)
- Python Version:** 2
- Autopilot Options:** Includes checkboxes for 'Enable autoscaling' and 'Terminate after [duration] minutes of inactivity'. The duration '120' is highlighted with a red box.
- Worker Type:** Standard\_DS3\_v2 (14.0 GB Memory, 4 Cores, 0.75 DBU) with Min Workers set to 2 and Max Workers set to 8.
- Driver Type:** Same as worker (14.0 GB Memory, 4 Cores, 0.75 DBU)

At the bottom, there's a link to 'Advanced Options'.

Fill in values for the following fields, and accept the default values for the other fields:

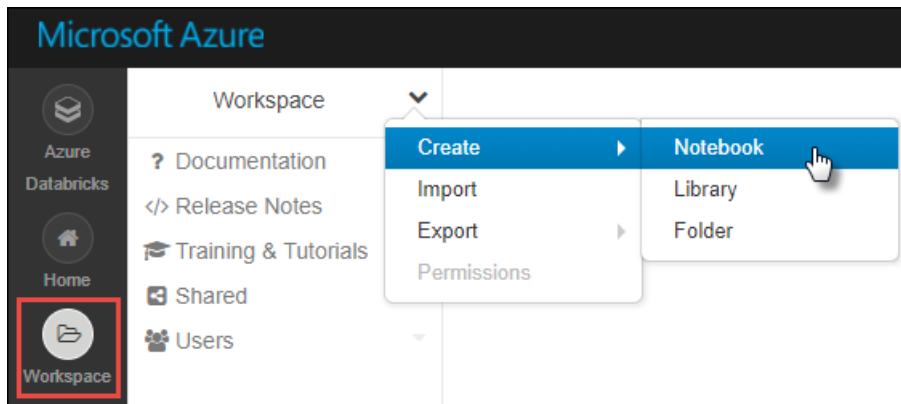
- Enter a name for the cluster.
  - Make sure you select the **Terminate after 120 minutes of inactivity** checkbox. Provide a duration (in minutes) to terminate the cluster, if the cluster is not being used.
4. Select **Create cluster**. Once the cluster is running, you can attach notebooks to the cluster and run Spark jobs.

For more information on creating clusters, see [Create a Spark cluster in Azure Databricks](#).

## Create storage account container

In this section, you create a notebook in Azure Databricks workspace and then run code snippets to configure the storage account.

- In the [Azure portal](#), go to the Azure Databricks workspace you created, and then select **Launch Workspace**.
- In the left pane, select **Workspace**. From the **Workspace** drop-down, select **Create > Notebook**.



3. In the **Create Notebook** dialog box, enter a name for the notebook. Select **Scala** as the language, and then select the Spark cluster that you created earlier.

**Create Notebook**

Name	mynotebook
Language	Scala
Cluster	mysparkcluster (42 GB, Running)

**Cancel** **Create**

Select **Create**.

4. Copy and paste the following code block into the first cell, but don't run this code yet.

```
spark.conf.set("fs.azure.account.auth.type.<storage-account-name>.dfs.core.windows.net", "OAuth")
spark.conf.set("fs.azure.account.oauth.provider.type.<storage-account-name>.dfs.core.windows.net",
"org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider")
spark.conf.set("fs.azure.account.oauth2.client.id.<storage-account-name>.dfs.core.windows.net", "
<appID>")
spark.conf.set("fs.azure.account.oauth2.client.secret.<storage-account-name>.dfs.core.windows.net", "
<password>")
spark.conf.set("fs.azure.account.oauth2.client.endpoint.<storage-account-name>.dfs.core.windows.net",
"https://login.microsoftonline.com/<tenant-id>/oauth2/token")
spark.conf.set("fs.azure.createRemoteFileSystemDuringInitialization", "true")
dbutils.fs.ls("abfss://<container-name>@<storage-account-name>.dfs.core.windows.net/")
spark.conf.set("fs.azure.createRemoteFileSystemDuringInitialization", "false")
```

5. In this code block, replace the `<storage-account-name>`, `<appID>`, `<password>`, and `<tenant-id>` placeholder values in this code block with the values that you collected when you created the service principal. Set the `<container-name>` placeholder value to whatever name you want to give the container.

6. Press the **SHIFT + ENTER** keys to run the code in this block.

## Ingest sample data

Before you begin with this section, you must complete the following prerequisites:

Enter the following code into a notebook cell:

```
%sh wget -P /tmp
https://raw.githubusercontent.com/Azure/usql/master/Examples/Samples/Data/json/radiowebsite/small_radio_json.json
```

In the cell, press **SHIFT + ENTER** to run the code.

Now in a new cell below this one, enter the following code, and replace the values that appear in brackets with the same values you used earlier:

```
dbutils.fs.cp("file:///tmp/small_radio_json.json", "abfss://<container-name>@<storage-account-name>.dfs.core.windows.net/")
```

In the cell, press **SHIFT + ENTER** to run the code.

## Run a Spark SQL Job

Perform the following tasks to run a Spark SQL job on the data.

1. Run a SQL statement to create a temporary table using data from the sample JSON data file, **small\_radio\_json.json**. In the following snippet, replace the placeholder values with your container name and storage account name. Using the notebook you created earlier, paste the snippet in a new code cell in the notebook, and then press **SHIFT + ENTER**.

```
%sql
DROP TABLE IF EXISTS radio_sample_data;
CREATE TABLE radio_sample_data
USING json
OPTIONS (
 path "abfss://<container-name>@<storage-account-name>.dfs.core.windows.net/small_radio_json.json"
)
```

Once the command successfully completes, you have all the data from the JSON file as a table in Databricks cluster.

The `%sql` language magic command enables you to run a SQL code from the notebook, even if the notebook is of another type. For more information, see [Mixing languages in a notebook](#).

2. Let's look at a snapshot of the sample JSON data to better understand the query that you run. Paste the following snippet in the code cell and press **SHIFT + ENTER**.

```
%sql
SELECT * from radio_sample_data
```

3. You see a tabular output like shown in the following screenshot (only some columns are shown):

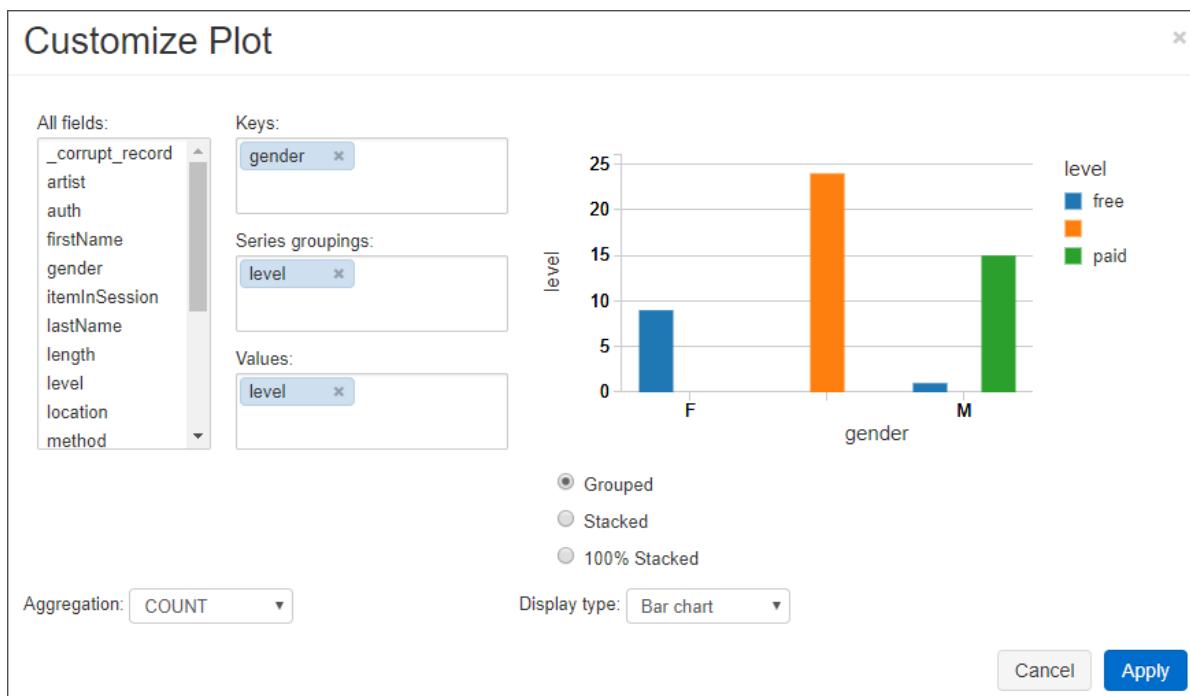
artist	auth	firstName	gender	itemInSession	lastName	length	level
El Arrebato	Logged In	Annalyse	F	2	Montgomery	234.57914	free
Creedence Clearwater Revival	Logged In	Dylann	M	9	Thomas	340.87138	paid
Gorillaz	Logged In	Liam	M	11	Watts	246.17751	paid
null	Logged In	Tess	F	0	Townsend	null	free
Otis Redding	Logged In	Margaux	F	2	Smith	135.57506	free

Among other details, the sample data captures the gender of the audience of a radio channel (column name, **gender**) and whether their subscription is free or paid (column name, **level**).

4. You now create a visual representation of this data to show for each gender, how many users have free accounts and how many are paid subscribers. From the bottom of the tabular output, click the **Bar chart** icon, and then click **Plot Options**.



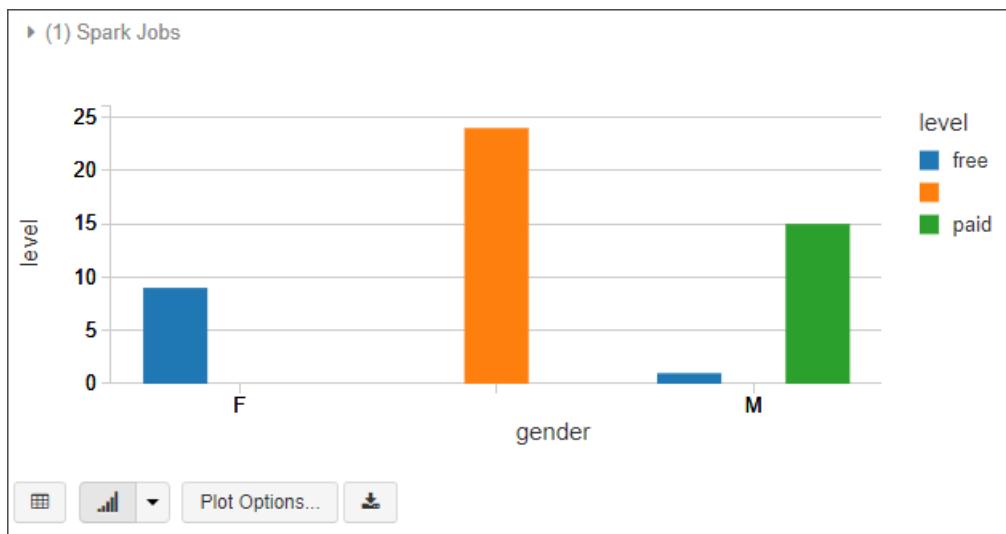
5. In **Customize Plot**, drag-and-drop values as shown in the screenshot.



- Set **Keys** to **gender**.
- Set **Series groupings** to **level**.
- Set **Values** to **level**.
- Set **Aggregation** to **COUNT**.

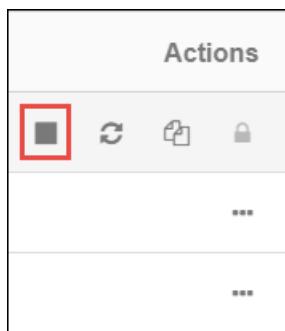
6. Click **Apply**.

7. The output shows the visual representation as depicted in the following screenshot:



## Clean up resources

Once you're finished with this article, you can terminate the cluster. From the Azure Databricks workspace, select **Clusters** and locate the cluster you want to terminate. Hover your mouse cursor over the ellipsis under **Actions** column, and select the **Terminate** icon.



If you do not manually terminate the cluster it automatically stops, provided you selected the **Terminate after \_\_\_\_ minutes of inactivity** checkbox while creating the cluster. If you set this option the cluster stops after it has been inactive for the designated amount of time.

## Next steps

In this article, you created a Spark cluster in Azure Databricks and ran a Spark job using data in a storage account with Data Lake Storage Gen2 enabled.

Advance to the next article to learn how to perform an ETL operation (extract, transform, and load data) using Azure Databricks.

### [Extract, transform, and load data using Azure Databricks.](#)

- To learn how to import data from other data sources into Azure Databricks, see [Spark data sources](#).
- To learn about other ways to access Azure Data Lake Storage Gen2 from an Azure Databricks workspace, see [Azure Data Lake Storage Gen2](#).

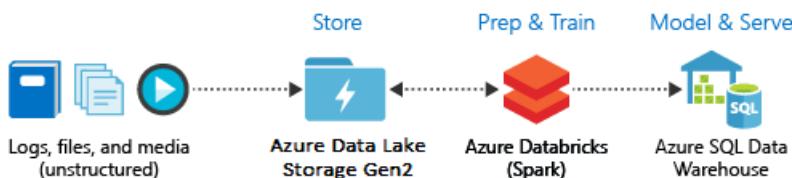
# Tutorial: Extract, transform, and load data by using Azure Databricks

4/14/2020 • 12 minutes to read • [Edit Online](#)

In this tutorial, you perform an ETL (extract, transform, and load data) operation by using Azure Databricks. You extract data from Azure Data Lake Storage Gen2 into Azure Databricks, run transformations on the data in Azure Databricks, and load the transformed data into Azure Synapse Analytics.

The steps in this tutorial use the Azure Synapse connector for Azure Databricks to transfer data to Azure Databricks. This connector, in turn, uses Azure Blob Storage as temporary storage for the data being transferred between an Azure Databricks cluster and Azure Synapse.

The following illustration shows the application flow:



This tutorial covers the following tasks:

- Create an Azure Databricks service.
- Create a Spark cluster in Azure Databricks.
- Create a file system in the Data Lake Storage Gen2 account.
- Upload sample data to the Azure Data Lake Storage Gen2 account.
- Create a service principal.
- Extract data from the Azure Data Lake Storage Gen2 account.
- Transform data in Azure Databricks.
- Load data into Azure Synapse.

If you don't have an Azure subscription, create a [free account](#) before you begin.

## NOTE

This tutorial cannot be carried out using **Azure Free Trial Subscription**. If you have a free account, go to your profile and change your subscription to **pay-as-you-go**. For more information, see [Azure free account](#). Then, [remove the spending limit](#), and [request a quota increase](#) for vCPUs in your region. When you create your Azure Databricks workspace, you can select the **Trial (Premium - 14-Days Free DBUs)** pricing tier to give the workspace access to free Premium Azure Databricks DBUs for 14 days.

## Prerequisites

Complete these tasks before you begin this tutorial:

- Create an Azure Synapse, create a server-level firewall rule, and connect to the server as a server admin. See [Quickstart: Create and query a Synapse SQL pool using the Azure portal](#).
- Create a master key for the Azure Synapse. See [Create a database master key](#).
- Create an Azure Blob storage account, and a container within it. Also, retrieve the access key to access the

storage account. See [Quickstart: Upload, download, and list blobs with the Azure portal](#).

- Create an Azure Data Lake Storage Gen2 storage account. See [Quickstart: Create an Azure Data Lake Storage Gen2 storage account](#).
- 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 in the scope of the Data Lake Storage Gen2 account. If you assign the role to the parent resource group or subscription, you'll receive permissions-related errors until those role assignments propagate to the storage account.  
If you'd prefer to use an access control list (ACL) to associate the service principal with a specific file or directory, reference [Access control in Azure Data Lake Storage Gen2](#).
- When performing the steps in the [Get values for signing in](#) section of the article, paste the tenant ID, app ID, and secret values into a text file.
- Sign in to the [Azure portal](#).

## Gather the information that you need

Make sure that you complete the prerequisites of this tutorial.

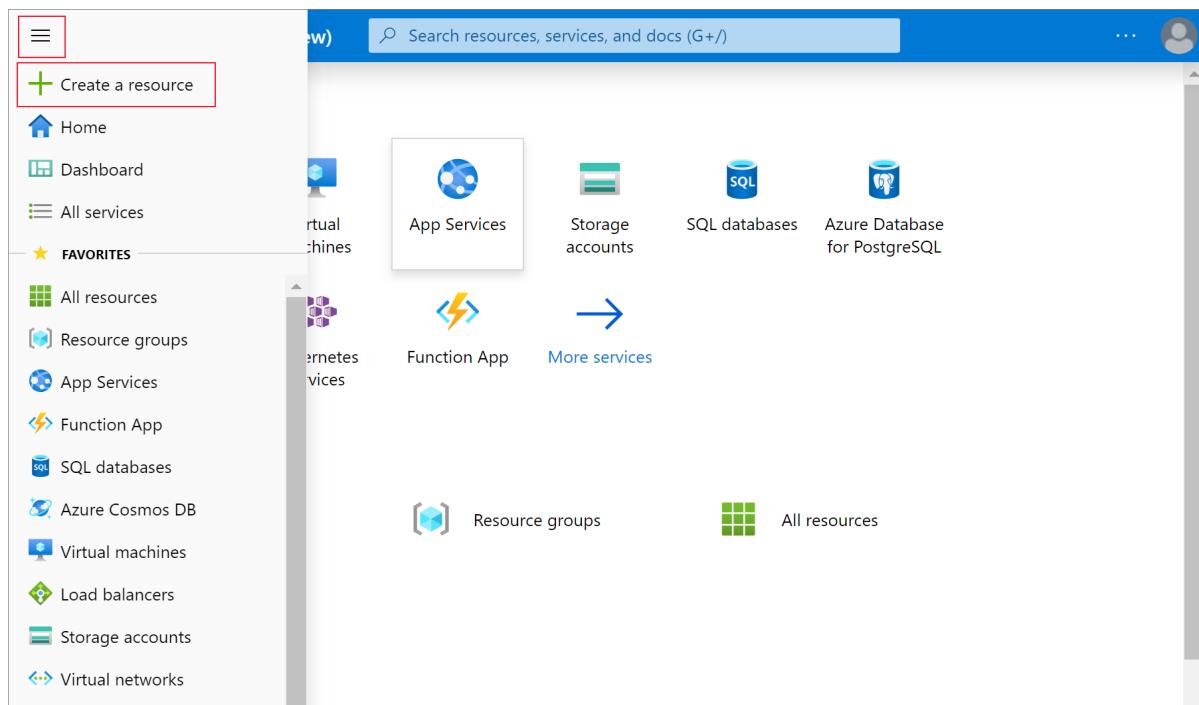
Before you begin, you should have these items of information:

- ✓ The database name, database server name, user name, and password of your Azure Synapse.
- ✓ The access key of your blob storage account.
- ✓ The name of your Data Lake Storage Gen2 storage account.
- ✓ The tenant ID of your subscription.
- ✓ The application ID of the app that you registered with Azure Active Directory (Azure AD).
- ✓ The authentication key for the app that you registered with Azure AD.

## Create an Azure Databricks service

In this section, you create an Azure Databricks service by using the Azure portal.

1. From the Azure portal menu, select **Create a resource**.



Then, select **Analytics** > **Azure Databricks**.

This screenshot shows the Azure Marketplace page. At the top, there's a search bar with the placeholder 'Search the Marketplace'. Below it, there are two tabs: 'Azure Marketplace' and 'See all'. On the left, there's a sidebar with categories: 'Get started', 'Recently created', 'AI + Machine Learning', 'Analytics' (which is highlighted with a red box), 'Blockchain', 'Compute', 'Containers', 'Databases', 'Developer Tools', 'DevOps', 'Identity', 'Integration', 'Internet of Things', 'Media', and 'Mixed Reality'. On the right, there's a 'Featured' section with several items: 'Azure Data Explorer' (with a blue icon), 'Azure HDInsight' (with a blue hexagonal icon), 'Data Lake Analytics' (with a purple icon), 'Stream Analytics job' (with a purple gear icon), 'Analysis Services' (with a dark purple icon), 'Azure Databricks' (with a red box around its icon), and 'Power BI Embedded' (with a yellow icon).

2. Under **Azure Databricks Service**, provide the following values to create a Databricks service:

PROPERTY	DESCRIPTION
Workspace name	Provide a name for your Databricks workspace.
Subscription	From the drop-down, select your Azure subscription.
Resource group	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> .
Location	Select <b>West US 2</b> . For other available regions, see <a href="#">Azure services available by region</a> .
Pricing Tier	Select <b>Standard</b> .

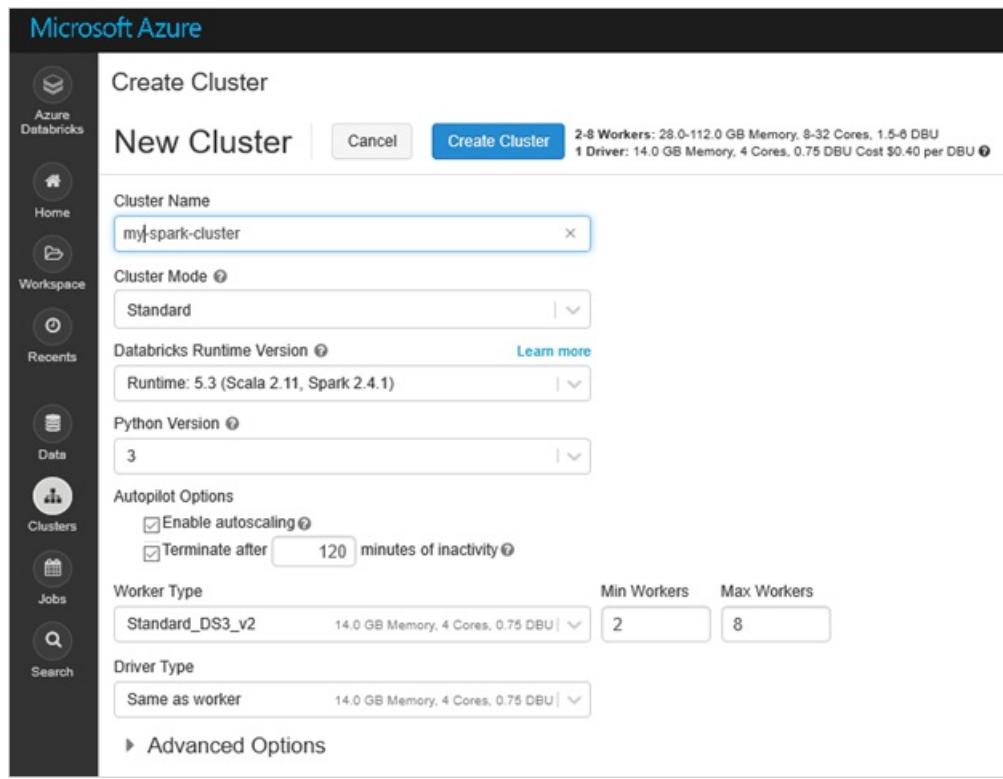
3. The account creation takes a few minutes. To monitor the operation status, view the progress bar at the top.
4. Select **Pin to dashboard** and then select **Create**.

## 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 portal. At the top, there's a logo and the text "Azure Databricks". Below it, under "Featured Notebooks", there are three cards: "Introduction to Apache Spark on Databricks" (Python icon), "Databricks for Data Scientists" (red server icon), and "Introduction to Structured Streaming" (Python icon). In the center, there's a "New" section with icons for Notebook, Job, Cluster (which is highlighted with a red box), Table, and Library. To the right of this is a "Documentation" section with links to "Databricks Guide", "Python, R, Scala, SQL", and "Importing Data". Further right is an "Open Recent" section which is currently empty. A note in this section says: "Recent files appear here as you work. Get started with the [welcome guide](#)".

3. In the **New cluster** page, provide the values to create a cluster.



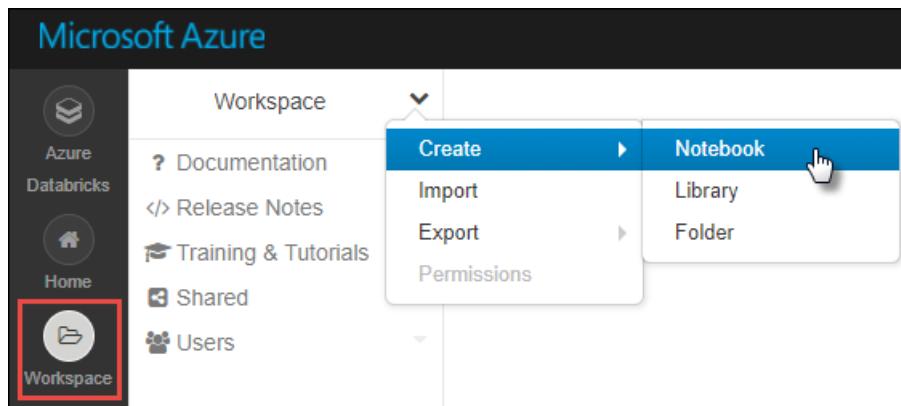
4. Fill in values for the following fields, and accept the default values for the other fields:

- Enter a name for the cluster.
- Make sure you select the **Terminate after \_\_ minutes of inactivity** check box. If the cluster isn't being used, provide a duration (in minutes) to terminate the cluster.
- Select **Create cluster**. After the cluster is running, you can attach notebooks to the cluster and run Spark jobs.

## Create a file system in the Azure Data Lake Storage Gen2 account

In this section, you create a notebook in Azure Databricks workspace and then run code snippets to configure the 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 **Scala** as the language, and then select the Spark cluster that you created earlier.

**Create Notebook**

Name	samplenotebook
Language	Scala
Cluster	mysparkcluster (42 GB, Running)
<input type="button" value="Cancel"/> <input type="button" value="Create"/>	

4. Select **Create**.
5. The following code block sets default service principal credentials for any ADLS Gen 2 account accessed in the Spark session. The second code block appends the account name to the setting to specify credentials for a specific ADLS Gen 2 account. Copy and paste either code block into the first cell of your Azure Databricks notebook.

### Session configuration

```
val appId = "<appID>"
val secret = "<secret>"
val tenantID = "<tenant-id>"

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", "<secret>")
spark.conf.set("fs.azure.account.oauth2.client.endpoint", "https://login.microsoftonline.com/<tenant-
id>/oauth2/token")
spark.conf.set("fs.azure.createRemoteFileSystemDuringInitialization", "true")
```

### Account configuration

```
val storageAccountName = "<storage-account-name>"
val appId = "<app-id>"
val secret = "<secret>"
val fileSystemName = "<file-system-name>"
val tenantID = "<tenant-id>"

spark.conf.set("fs.azure.account.auth.type." + storageAccountName + ".dfs.core.windows.net", "OAuth")
spark.conf.set("fs.azure.account.oauth.provider.type." + storageAccountName + ".dfs.core.windows.net",
"org.apache.hadoop.fs.azurebfs.oauth2.ClientCredsTokenProvider")
spark.conf.set("fs.azure.account.oauth2.client.id." + storageAccountName + ".dfs.core.windows.net", "" +
appId + "")
spark.conf.set("fs.azure.account.oauth2.client.secret." + storageAccountName + ".dfs.core.windows.net",
"" + secret + "")
spark.conf.set("fs.azure.account.oauth2.client.endpoint." + storageAccountName +
.dfs.core.windows.net", "https://login.microsoftonline.com/" + tenantID + "/oauth2/token")
spark.conf.set("fs.azure.createRemoteFileSystemDuringInitialization", "true")
dutils.fs.ls("abfss://" + fileSystemName + "@" + storageAccountName + ".dfs.core.windows.net/")
spark.conf.set("fs.azure.createRemoteFileSystemDuringInitialization", "false")
```

6. In this code block, replace the `<app-id>`, `<secret>`, `<tenant-id>`, 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 `<file-system-name>` placeholder value with whatever name you want to give the file system.

- The `<app-id>`, and `<secret>` are from the app that you registered with active directory as part of creating a service principal.
- The `<tenant-id>` is from your subscription.
- The `<storage-account-name>` is the name of your Azure Data Lake Storage Gen2 storage account.

7. Press the **SHIFT + ENTER** keys to run the code in this block.

## Ingest sample data into the Azure Data Lake Storage Gen2 account

Before you begin with this section, you must complete the following prerequisites:

Enter the following code into a notebook cell:

```
%sh wget -P /tmp
https://raw.githubusercontent.com/Azure/usql/master/Examples/Samples/Data/json/radiowebsite/small_radio_json.json
```

In the cell, press **SHIFT + ENTER** to run the code.

Now in a new cell below this one, enter the following code, and replace the values that appear in brackets with the same values you used earlier:

```
dbutils.fs.cp("file:///tmp/small_radio_json.json", "abfss://" + fileSystemName + "@" + storageAccountName +
".dfs.core.windows.net/")
```

In the cell, press **SHIFT + ENTER** to run the code.

## Extract data from the Azure Data Lake Storage Gen2 account

1. You can now load the sample json file as a data frame in Azure Databricks. Paste the following code in a new cell. Replace the placeholders shown in brackets with your values.

```
val df = spark.read.json("abfss://" + fileSystemName + "@" + storageAccountName +
".dfs.core.windows.net/small_radio_json.json")
```

2. Press the **SHIFT + ENTER** keys to run the code in this block.

3. Run the following code to see the contents of the data frame:

```
df.show()
```

You see an output similar to the following snippet:

```

+-----+-----+-----+-----+-----+-----+
| artist| auth|firstName|gender|itemInSession| lastName| length| level|
|location|method| page| registration|sessionId| song|status| ts|userId|
+-----+-----+-----+-----+-----+-----+
| El Arrebato | Logged In| Annalyse| F| 2|Montgomery|234.57914| free | Killeen-
Temple, TX| PUT|NextSong|1384448062332| 1879|Quiero Quererte Q...| 200|1409318650332| 309|
| Creedence Clearwa...|Logged In| Dylann| M| 9| Thomas|340.87138| paid |
Anchorage, AK| PUT|NextSong|1400723739332| 10| Born To Move| 200|1409318653332| 11|
| Gorillaz | Logged In| Liam| M| 11| Watts|246.17751| paid |New York-
Newark-J...| PUT|NextSong|1406279422332| 2047| DARE| 200|1409318685332| 201|
...
...

```

You have now extracted the data from Azure Data Lake Storage Gen2 into Azure Databricks.

## Transform data in Azure Databricks

The raw sample data **small\_radio\_json.json** file captures the audience for a radio station and has a variety of columns. In this section, you transform the data to only retrieve specific columns from the dataset.

1. First, retrieve only the columns **firstName**, **lastName**, **gender**, **location**, and **level** from the dataframe that you created.

```

val specificColumnsDf = df.select("firstname", "lastname", "gender", "location", "level")
specificColumnsDf.show()

```

You receive output as shown in the following snippet:

```

+-----+-----+-----+-----+
|firstname| lastname|gender| location|level|
+-----+-----+-----+-----+
| Annalyse|Montgomery| F| Killeen-Temple, TX| free|
| Dylann| Thomas| M| Anchorage, AK| paid|
| Liam| Watts| M|New York-Newark-J...| paid|
| Tess| Townsend| F|Nashville-Davidso...| free|
| Margaux| Smith| F|Atlanta-Sandy Spr...| free|
| Alan| Morse| M|Chicago-Napervill...| paid|
| Gabriella| Shelton| F|San Jose-Sunnyval...| free|
| Elijah| Williams| M|Detroit-Warren-De...| paid|
| Margaux| Smith| F|Atlanta-Sandy Spr...| free|
| Tess| Townsend| F|Nashville-Davidso...| free|
| Alan| Morse| M|Chicago-Napervill...| paid|
| Liam| Watts| M|New York-Newark-J...| paid|
| Liam| Watts| M|New York-Newark-J...| paid|
| Dylann| Thomas| M| Anchorage, AK| paid|
| Alan| Morse| M|Chicago-Napervill...| paid|
| Elijah| Williams| M|Detroit-Warren-De...| paid|
| Margaux| Smith| F|Atlanta-Sandy Spr...| free|
| Alan| Morse| M|Chicago-Napervill...| paid|
| Dylann| Thomas| M| Anchorage, AK| paid|
| Margaux| Smith| F|Atlanta-Sandy Spr...| free|

```

2. You can further transform this data to rename the column **level** to **subscription\_type**.

```

val renamedColumnsDF = specificColumnsDf.withColumnRenamed("level", "subscription_type")
renamedColumnsDF.show()

```

You receive output as shown in the following snippet.

firstname	lastname	gender	location	subscription_type
Annalyse	Montgomery	F	Killeen-Temple, TX	free
Dylann	Thomas	M	Anchorage, AK	paid
Liam	Watts	M	New York-Newark-J...	paid
Tess	Townsend	F	Nashville-Davidso...	free
Margaux	Smith	F	Atlanta-Sandy Spr...	free
Alan	Morse	M	Chicago-Napervill...	paid
Gabriella	Shelton	F	San Jose-Sunnyval...	free
Elijah	Williams	M	Detroit-Warren-De...	paid
Margaux	Smith	F	Atlanta-Sandy Spr...	free
Tess	Townsend	F	Nashville-Davidso...	free
Alan	Morse	M	Chicago-Napervill...	paid
Liam	Watts	M	New York-Newark-J...	paid
Liam	Watts	M	New York-Newark-J...	paid
Dylann	Thomas	M	Anchorage, AK	paid
Alan	Morse	M	Chicago-Napervill...	paid
Elijah	Williams	M	Detroit-Warren-De...	paid
Margaux	Smith	F	Atlanta-Sandy Spr...	free
Alan	Morse	M	Chicago-Napervill...	paid
Dylann	Thomas	M	Anchorage, AK	paid
Margaux	Smith	F	Atlanta-Sandy Spr...	free

## Load data into Azure Synapse

In this section, you upload the transformed data into Azure Synapse. You use the Azure Synapse connector for Azure Databricks to directly upload a dataframe as a table in a Synapse Spark pool.

As mentioned earlier, the Azure Synapse connector uses Azure Blob storage as temporary storage to upload data between Azure Databricks and Azure Synapse. So, you start by providing the configuration to connect to the storage account. You must already have already created the account as part of the prerequisites for this article.

1. Provide the configuration to access the Azure Storage account from Azure Databricks.

```
val blobStorage = "<blob-storage-account-name>.blob.core.windows.net"
val blobContainer = "<blob-container-name>"
val blobAccessKey = "<access-key>"
```

2. Specify a temporary folder to use while moving data between Azure Databricks and Azure Synapse.

```
val tempDir = "wasbs://" + blobContainer + "@" + blobStorage + "/tempDirs"
```

3. Run the following snippet to store Azure Blob storage access keys in the configuration. This action ensures that you don't have to keep the access key in the notebook in plain text.

```
val acntInfo = "fs.azure.account.key." + blobStorage
sc.hadoopConfiguration.set(acntInfo, blobAccessKey)
```

4. Provide the values to connect to the Azure Synapse instance. You must have created an Azure Synapse Analytics service as a prerequisite. Use the fully qualified server name for **dwServer**. For example, <servername>.database.windows.net .

```

//Azure Synapse related settings
val dwDatabase = "<database-name>"
val dwServer = "<database-server-name>"
val dwUser = "<user-name>"
val dwPass = "<password>"
val dwJdbcPort = "1433"
val dwJdbcExtraOptions =
"encrypt=true;trustServerCertificate=true;hostNameInCertificate=*.database.windows.net;loginTimeout=30;"
val sqlDwUrl = "jdbc:sqlserver://" + dwServer + ":" + dwJdbcPort + ";database=" + dwDatabase + ";user="
+ dwUser + ";password=" + dwPass + ";" + dwJdbcExtraOptions
val sqlDwUrlSmall = "jdbc:sqlserver://" + dwServer + ":" + dwJdbcPort + ";database=" + dwDatabase +
";user=" + dwUser + ";password=" + dwPass

```

- Run the following snippet to load the transformed dataframe, `renamedColumnsDF`, as a table in Azure Synapse. This snippet creates a table called `SampleTable` in the SQL database.

```

spark.conf.set(
 "spark.sql.parquet.writeLegacyFormat",
 "true")

renamedColumnsDF.write.format("com.databricks.spark.sqldw").option("url",
sqlDwUrlSmall).option("dbtable", "SampleTable") .option(
"forward_spark_azure_storage_credentials","True").option("tempdir", tempDir).mode("overwrite").save()

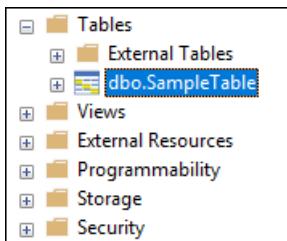
```

#### NOTE

This sample uses the `forward_spark_azure_storage_credentials` flag, which causes Azure Synapse to access data from blob storage using an Access Key. This is the only supported method of authentication.

If your Azure Blob Storage is restricted to select virtual networks, Azure Synapse requires [Managed Service Identity](#) instead of [Access Keys](#). This will cause the error "This request is not authorized to perform this operation."

- Connect to the SQL database and verify that you see a database named `SampleTable`.

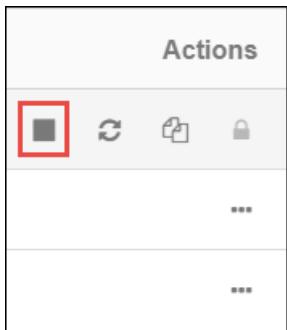


- Run a select query to verify the contents of the table. The table should have the same data as the `renamedColumnsDF` dataframe.

	firstname	lastname	gender	location	subscription_type
1	Annalise	Montgomery	F	Killeen-Temple, TX	free
2	Dylann	Thomas	M	Anchorage, AK	paid
3	Liam	Watts	M	New York-Newark-Jersey City, NY-NJ-PA	paid
4	Tess	Townsend	F	Nashville-Davidson-Murfreesboro-Franklin, TN	free
5	Margaux	Smith	F	Atlanta-Sandy Springs-Roswell, GA	free
6	Alan	Morse	M	Chicago-Naperville-Elgin, IL-IN-WI	paid
7	Gabriella	Shelton	F	San Jose-Sunnyvale-Santa Clara, CA	free
8	Elijah	Williams	M	Detroit-Warren-Dearborn, MI	paid
9	Margaux	Smith	F	Atlanta-Sandy Springs-Roswell, GA	free
10	Tess	Townsend	F	Nashville-Davidson-Murfreesboro-Franklin, TN	free
11	Alan	Morse	M	Chicago-Naperville-Elgin, IL-IN-WI	paid
12	Liam	Watts	M	New York-Newark-Jersey City, NY-NJ-PA	paid
13	Liam	Watts	M	New York-Newark-Jersey City, NY-NJ-PA	paid
14	Dylann	Thomas	M	Anchorage, AK	paid
15	Alan	Morse	M	Chicago-Naperville-Elgin, IL-IN-WI	paid
16	Elijah	Williams	M	Detroit-Warren-Dearborn, MI	paid
17	Margaux	Smith	F	Atlanta-Sandy Springs-Roswell, GA	free
18	Alan	Morse	M	Chicago-Naperville-Elgin, IL-IN-WI	paid
19	Dylann	Thomas	M	Anchorage, AK	paid
20	Margaux	Smith	F	Atlanta-Sandy Springs-Roswell, GA	free

## Clean up resources

After you finish the tutorial, you can terminate the cluster. From the Azure Databricks workspace, select **Clusters** on the left. For the cluster to terminate, under **Actions**, point to the ellipsis (...) and select the **Terminate** icon.



If you don't manually terminate the cluster, it automatically stops, provided you selected the **Terminate after \_\_\_\_ minutes of inactivity** check box when you created the cluster. In such a case, the cluster automatically stops if it's been inactive for the specified time.

## Next steps

In this tutorial, you learned how to:

- Create an Azure Databricks service
- Create a Spark cluster in Azure Databricks
- Create a notebook in Azure Databricks
- Extract data from a Data Lake Storage Gen2 account
- Transform data in Azure Databricks
- Load data into Azure Synapse

Advance to the next tutorial to learn about streaming real-time data into Azure Databricks using Azure Event Hubs.

[Stream data into Azure Databricks using Event Hubs](#)

# Tutorial: Azure Data Lake Storage Gen2, Azure Databricks & Spark

4/14/2020 • 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 an Azure Data Lake Storage Gen2 account](#).
- 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. 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.

## 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 Compute, Storage, and Analytics. Under 'Analytics', the 'Azure Databricks' service is listed and highlighted with a red box. Other services shown include Power BI Embedded, SQL Data Warehouse, Data Lake Storage Gen1, and Data Factory.

2. Under **Azure Databricks Service**, provide the following values to create a Databricks service:

PROPERTY	DESCRIPTION
Workspace name	Provide a name for your Databricks workspace.
Subscription	From the drop-down, select your Azure subscription.
Resource group	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> .
Location	Select <b>West US 2</b> . For other available regions, see <a href="#">Azure services available by region</a> .
Pricing Tier	Select <b>Standard</b> .

Home > New > Azure Databricks Service

## Azure Databricks Service

**\* Workspace name**  
mydatabricksaws

**\* Subscription**

**\* Resource group i**  
 Create new  Use existing  
mydatabricksresgrp

**\* Location**  
West US 2

**\* Pricing Tier ( View full pricing details )**

Standard (Apache Spark, Secure with Azure AD)  
Premium (+ Role-based access controls)

Pin to dashboard

**Create**   [Automation options](#)

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**.

Featured Notebooks

Introduction to Apache Spark on Databricks   Databricks for Data Scientists   Introduction to Structured Streaming

New

- Notebook
- Job
- Cluster
- Table
- Library

Documentation

- Databricks Guide
- Python, R, Scala, SQL
- Importing Data

Open Recent

Recent files appear here as you work. Get started with the [welcome guide](#).

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
```

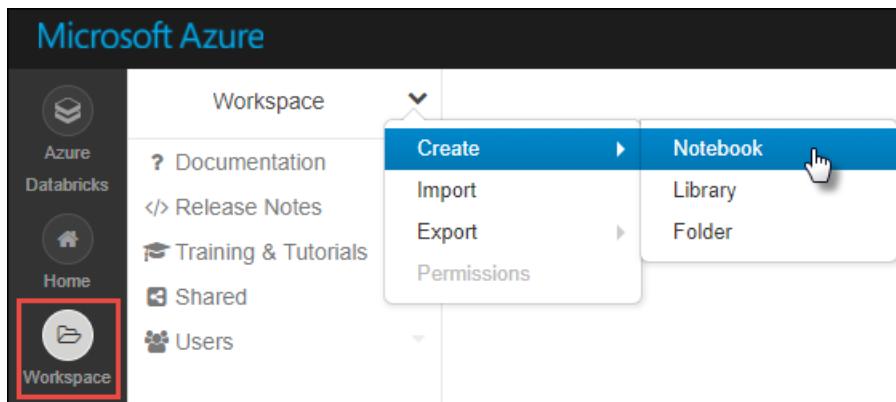
- Replace the <csv-folder-path> placeholder value with the path to the .csv file.

- 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

11/21/2019 • 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 an 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 an 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 an Azure SQL database as a destination data store. If you don't have a SQL database, see [Create an 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 the flight data

1. Browse to [Research and Innovative Technology Administration, Bureau of Transportation Statistics](#).
2. On the page, select the following values:

NAME	VALUE
Filter Year	2013
Filter Period	January
Fields	Year, FlightDate, Reporting_Airline, IATA_CODE_Reported_Airline, Flight_Number_Reported_Airline, OriginAirportID, Origin, OriginCityName, OriginState, DestAirportID, Dest, DestCityName, DestState, DepDelayMinutes, ArrDelay, ArrDelayMinutes, CarrierDelay, WeatherDelay, NASDelay, SecurityDelay, LateAircraftDelay.

Clear all other fields.

3. Select **Download**. You get a .zip file with the data fields you selected.

## Extract and upload the data

In this section, you'll upload data to your HDInsight cluster and then copy that data to your Data Lake Storage Gen2 account.

1. 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: .
```

2. 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
```

3. Use the following command to unzip the .zip file:

```
unzip <file-name>.zip
```

The command extracts a .csv file.

4. 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.

5. 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
```

6. 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  **Y** 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

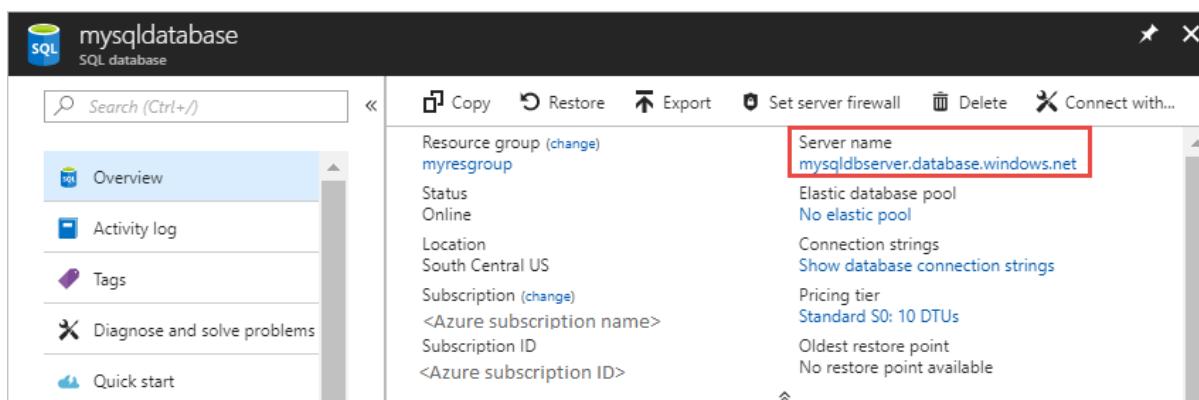
`abfs://<container-name>@<storage-account-name>.dfs.core.windows.net/tutorials/flightdelays/output`. Later, Sqoop reads the data from this location and exports it to Azure SQL Database.

7. To exit Beeline, enter `!quit` at the prompt.

## Create a SQL database table

You need the server name from your 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
```

- After the installation completes, use the following command to connect to the SQL Database server.

```
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 SQL Database 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>
```

- 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
```

- When the `GO` statement is entered, the previous statements are evaluated.

The query creates a table named `delays`, which has a clustered index.

- 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

- 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/flightdelays/output`. In this section, you use Sqoop to export the data from

`abfs://<container-name>@<storage-account-name>.dfs.core.windows.net/tutorials/flightdelays/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 **hivesampletable** 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

3/4/2020 • 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 an Azure Data Lake Storage Gen2 account](#).

- 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.

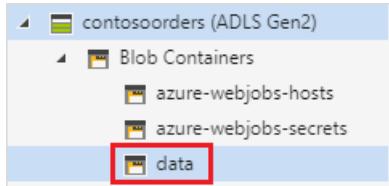
- ✓ When performing the steps in the [Get values for signing in](#) section of the article, paste the tenant ID, app ID, and password values into a text file. You'll need those values later.

## 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**.

The screenshot shows the Microsoft Azure portal's 'New' blade. On the left, the navigation menu includes 'Create a resource', 'All services', and a 'FAVORITES' section with links to Storage accounts, Dashboard, All resources, Resource groups, App Services, Function Apps, SQL databases, Azure Cosmos DB, Virtual machines, Load balancers, and Virtual networks. The main area displays the 'Azure Marketplace' tab under the 'Featured' heading. A search bar at the top says 'Search the Marketplace'. Below it, there are several service cards: HDInsight (Quickstart tutorial), Data Lake Analytics (Quickstart tutorial), Stream Analytics job (Quickstart tutorial), Analysis Services (Quickstart tutorial), Azure Databricks (Quickstart tutorial), Power BI Embedded (Quickstart tutorial), Integration, Internet of Things, Web, Storage, Mobile, Containers, Databases, and Analytics. The 'Analytics' and 'Azure Databricks' cards are both highlighted with red boxes.

- Under Azure Databricks Service, provide the values to create a Databricks workspace.

The screenshot shows the 'Azure Databricks Service' creation blade. It includes the following fields:

- \* Workspace name: contoso-orders
- \* Subscription: contoso
- \* Resource group: Use existing (contoso)
- \* Location: West US
- \* Pricing Tier: Standard (Apache Spark, Secure with Azur...)
- Deploy Azure Databricks workspace in your Virtual Network (preview): No

At the bottom, there are 'Create' and 'Automation options' buttons.

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, under "Featured Notebooks", are three cards: "Introduction to Apache Spark on Databricks" (Python icon), "Databricks for Data Scientists" (red bars icon), and "Introduction to Structured Streaming" (Python icon). The main area is divided into three sections: "New" (with "Cluster" highlighted and a red box around it), "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.

The screenshot shows the "Create Cluster" page in the Microsoft Azure portal. On the left, there's a sidebar with icons for Home, Workspace, Recents, Data, Clusters, Jobs, and Search. The main area has a title "Create Cluster" and a sub-section "New Cluster". It includes fields for "Cluster Name" (set to "customer-order-cluster" with a red box around it), "Cluster Mode" (set to "Standard"), "Pool" (set to "None"), "Databricks Runtime Version" (set to "Runtime: 5.3 (Scala 2.11, Spark 2.4.1)"), "Python Version" (set to "3"), and "Autopilot Options" (with "Enable autoscaling" checked and "Terminate after 120 minutes of inactivity" checked, both with a red box around them). At the bottom, there are sections for "Worker Type" (set to "Standard\_DS3\_v2") and "Driver Type" (set to "Same as worker"), and a link to "Advanced Options". A note at the top right says "2-8 Workers: 28.0-112.0 GB Memory, 8-32 Cores, 0.75 DBU" and "1 Driver: 14.0 GB Memory, 4 Cores, 0.75 DBU".

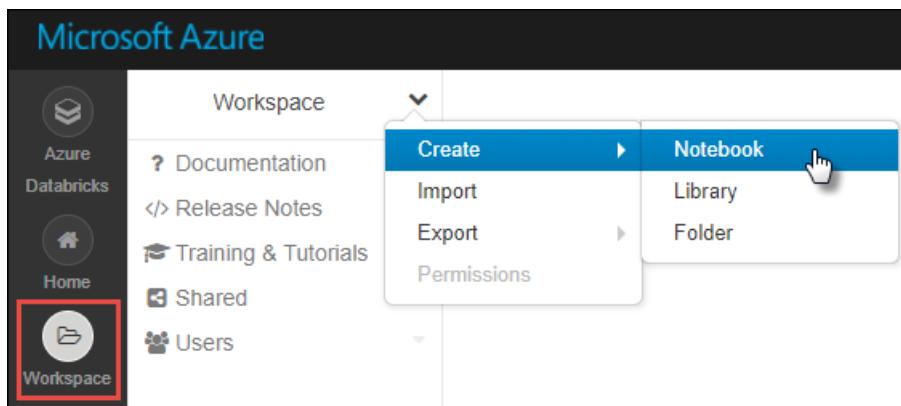
Accept all other default values other than the following:

- 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.

A screenshot of the 'Create Notebook' dialog box. It has three input fields: 'Name' with the value 'configure-customer-table', 'Language' set to 'Python', and 'Cluster' set to 'customer-order-cluster'. At the bottom are two buttons: 'Cancel' and a blue 'Create' button.

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.

- After this code block successfully runs, remove this code block from your notebook.

#### Add code that inserts rows into the Databricks Delta table

- 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.

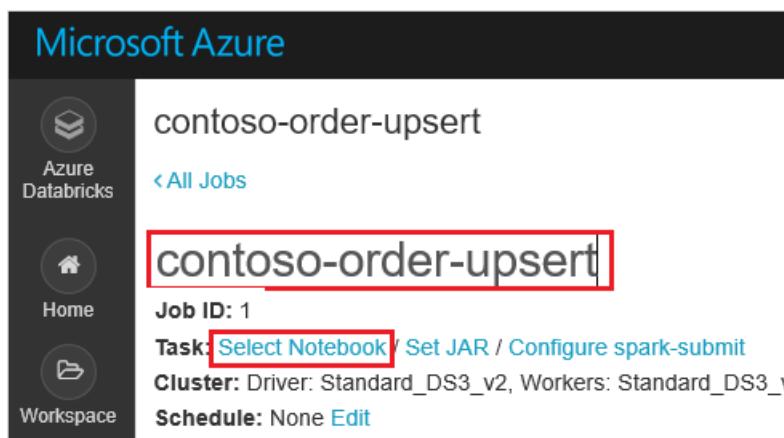
- Add the following code to merge the contents of the temporary table view with the Databricks Delta table.

```
%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.

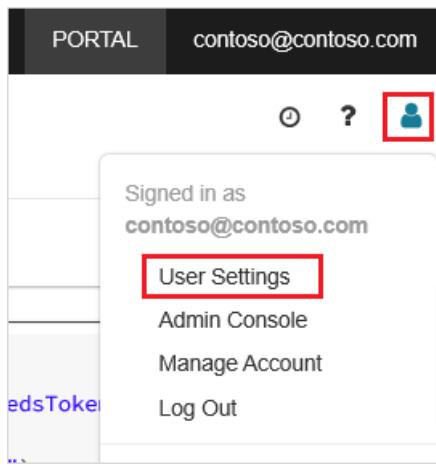
1. Click **Jobs**.
2. In the **Jobs** page, click **Create Job**.
3. Give the job a name, and then choose the `upsert-order-data` workbook.



## Create an Azure Function

Create an Azure Function that runs the Job.

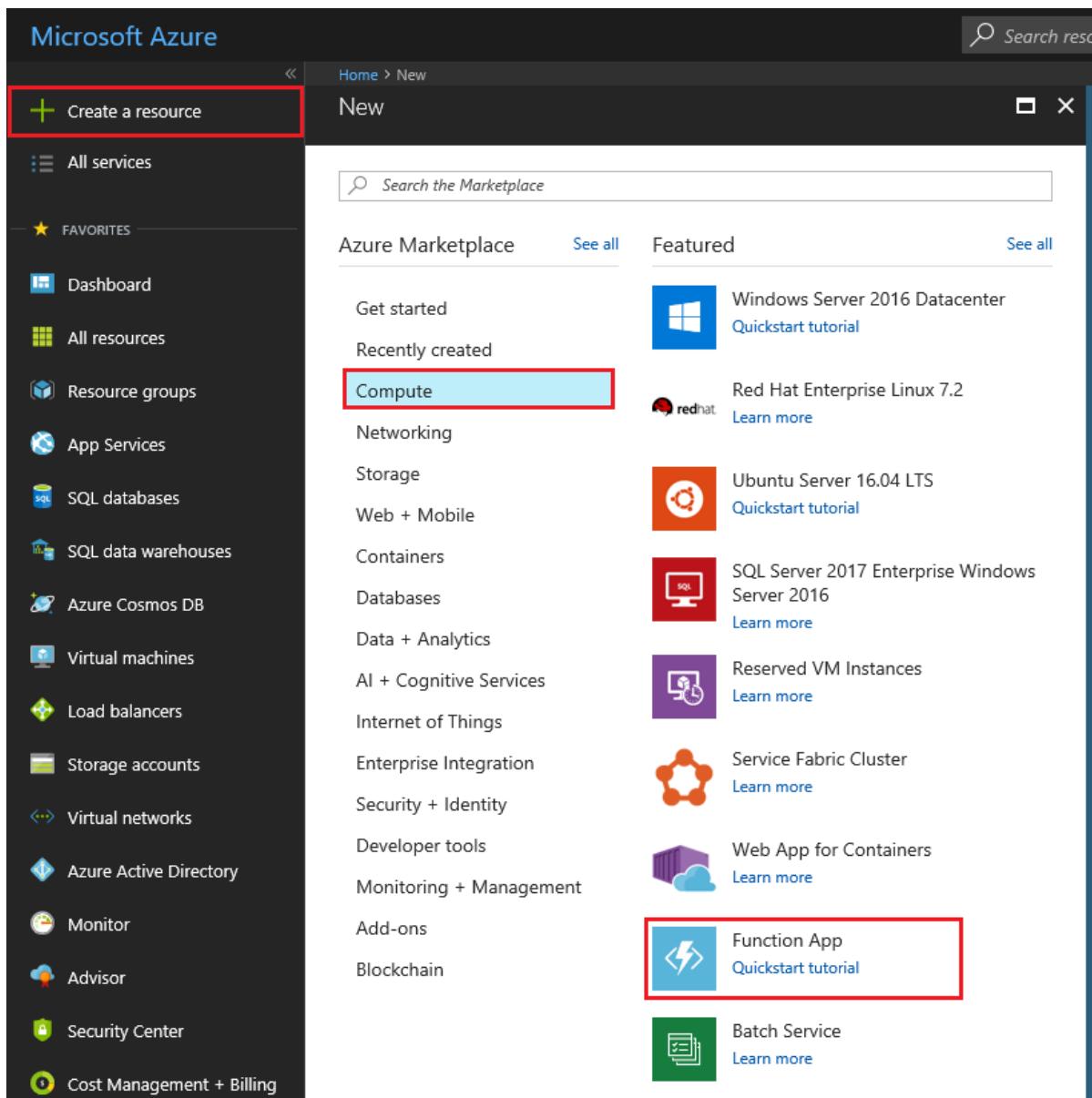
1. In the upper corner of the Databricks workspace, choose the people icon, and then choose **User settings**.



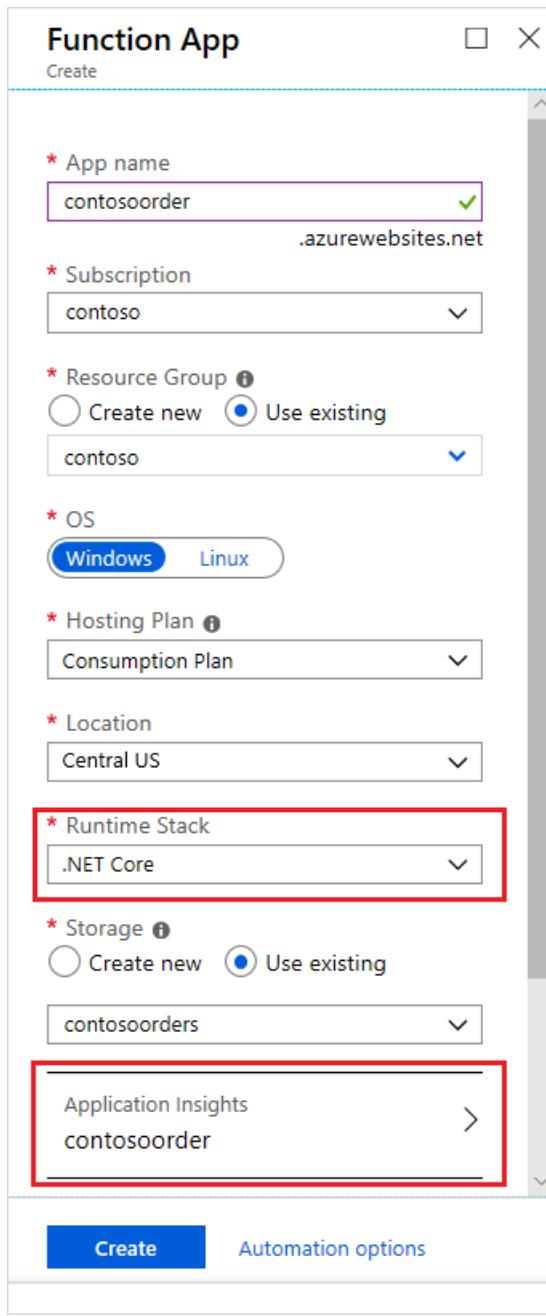
2. Click the **Generate new token** button, and then click the **Generate** button.

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.



5. In the **Overview** page of the Function App, click **Configuration**.

6. In the Application Settings page, choose the New application setting button to add each setting.

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 for a Function App named "contsoorder". On the left, there's a sidebar with a search bar containing "contsoorder" and a "New function" button. Below the search bar are sections for "Function Apps", "contsoorder" (expanded), "Functions" (selected and highlighted in blue), "Proxies", and "Slots (preview)". The main area is titled "Functions" and contains a search bar labeled "Search functions". A table with columns "NAME" and "STATUS" shows "No results".

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://{}{}/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.

The screenshot shows the Azure Functions portal interface. On the left, there's a sidebar with a search bar containing "contoso-order", a dropdown menu for "ADLstorePMTeam", and a "Function Apps" section with "Function Apps" and "contoso-order" listed. The main area has a title "contoso-order - UpstOrder" and a sub-section "Function Apps". Below that is a code editor window with the file name "run.csx". The code in the editor is:

```

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

```

At the top of the code editor, there are buttons for "Save", "Run", and "Add Event Grid subscription". The "Add Event Grid subscription" button is highlighted with a red border.

2. In the **Create Event Subscription** page, name the subscription, and then use the fields in the page to select your storage account.

The screenshot shows the "Create Event Subscription" page under the "Event Grid" section. At the top, there are tabs for "Basic", "Filters", and "Additional Features", with "Basic" selected. The main area is divided into several sections:

- EVENT SUBSCRIPTION DETAILS**: Fields for "Name" (contoso-order-event-subscription) and "Event Schema" (Event Grid Schema).
- TOPIC DETAILS**: A section for picking a topic resource. It includes "Topic Types" (Storage Accounts), "Subscription" (contoso), "Resource Group" (contoso), and "Resource" (contosoorders). The "Resource" field is highlighted with a red border.
- EVENT TYPES**: A section for picking event types. It includes a "Filter to Event Types" dropdown set to "2 selected".
- ENDPOINT DETAILS**: A section with a "Create" button.

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.

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 interface. 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 one job: 'contoso-order-upsert' (Job ID 1), created by 'consto employee' for the task 'upsert-order-data'. The 'contoso-order-upsert' cell is highlighted with a red box.

When the job completes, you'll see a completion status.

The screenshot shows a list of completed jobs in the past 60 days. It includes columns for Run, Run ID, Start Time, Launched, Duration, Spark, and Status. One run is listed: 'Run 10' (Run ID 10) started on 2019-08-12 at 14:44:44 PDT, launched manually, took 5m 34s, and has a status of 'Succeeded'. The 'Status' column is highlighted with a red box.

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 and a table viewer. The command line has the number '1' and the query '%sql select \* from customer\_data'. The table viewer shows a table with columns: InvoiceNo, StockCode, Description, Quantity, InvoiceDate, UnitPrice, CustomerID, and Country. Two rows are visible: one for a 'WHITE HANGING HEART T-LIGHT HOLDER' and one for an 'EverGlow Single'. The second row is highlighted with a red box. At the bottom, it says 'Command took 1.31 seconds -- by normesta@microsoft.com at 8/16/2019, 12:37:58 PM on customer-order-cluster'.

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.

A screenshot of a Jupyter Notebook cell. The cell contains the command `%sql select * from customer_data`. Below the cell, the output shows a table with 2 rows and 9 columns. The columns are: InvoiceNo, StockCode, Description, Quantity, InvoiceDate, UnitPrice, CustomerID, and Country. The first row has an InvoiceNo of 536365, StockCode of 85123A, Description of 'WHITE HANGING HEART T-LIGHT HOLDER', Quantity of 6, InvoiceDate of 12/1/2010 8:26, UnitPrice of 2.55, CustomerID of 17850, and Country of United Kingdom. The second row has an InvoiceNo of 536371, StockCode of 99999, Description of 'EverGlow Single', Quantity of 22 (which is highlighted with a red box), InvoiceDate of 1/1/2018 9:01, UnitPrice of 33.85, CustomerID of 20993, and Country of Sierra Leone. At the bottom of the output, it says "Command took 5.59 seconds -- by normesta@microsoft.com at 8/21/2019, 1:04:28 PM on customer-order-cluster".

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

## 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](#)

# Azure Storage samples using v12 .NET client libraries

4/9/2020 • 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](#)
- All other languages: [Azure Storage samples](#)

# Azure Storage samples using v12 Java client libraries

2/20/2020 • 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](#)

[Create a file using a Data Lake file client](#)

[Append data to a file](#)

[Download 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](#)
- All other languages: [Azure Storage samples](#)

# Azure Storage samples using v12 Python client libraries

2/20/2020 • 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](#)
- All other languages: [Azure Storage samples](#)



# Azure Storage samples using v12 JavaScript client libraries

2/20/2020 • 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](#)
- All other languages: [Azure Storage samples](#)

# Using Azure Data Lake Storage Gen2 for big data requirements

2/16/2020 • 6 minutes to read • [Edit Online](#)

There are four key stages in big data processing:

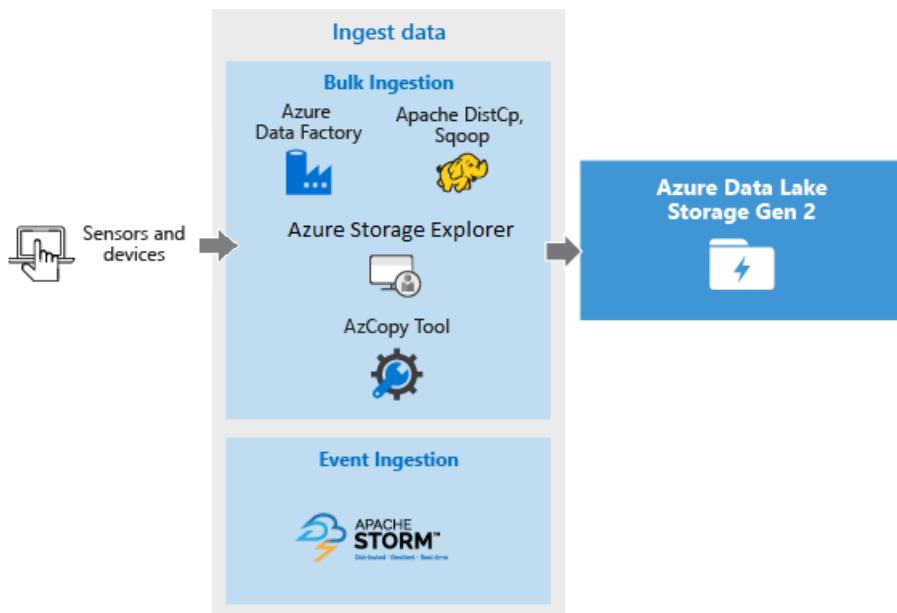
- Ingesting large amounts of data into a data store, at real-time or in batches
- Processing the data
- Downloading the data
- Visualizing the data

This article highlights the options and tools for each processing phase.

For a complete list of Azure services that you can use with Azure Data Lake Storage Gen2, see [Integrate Azure Data Lake Storage with Azure services](#)

## Ingest the data into Data Lake Storage Gen2

This section highlights the different sources of data and the different ways in which that data can be ingested into a Data Lake Storage Gen2 account.



### Ad hoc data

This represents smaller data sets that are used for prototyping a big data application. There are different ways of ingesting ad hoc data depending on the source of the data.

Here's a list of tools that you can use to ingest ad hoc data.

DATA SOURCE

INGEST IT USING

DATA SOURCE	INGEST IT USING
Local computer	Azure PowerShell
	Azure CLI
	Storage Explorer
	AzCopy tool
Azure Storage Blob	Azure Data Factory
	AzCopy tool
	DistCp running on HDInsight cluster

### Streamed data

This represents data that can be generated by various sources such as applications, devices, sensors, etc. This data can be ingested into Data Lake Storage Gen2 by a variety of tools. These tools will usually capture and process the data on an event-by-event basis in real-time, and then write the events in batches into Data Lake Storage Gen2 so that they can be further processed.

Here's a list of tools that you can use to ingest streamed data.

TOOL	GUIDANCE
Azure Stream Analytics	<a href="#">Quickstart: Create a Stream Analytics job by using the Azure portal</a> <a href="#">Egress to Azure Data Lake Gen2</a>
Azure HDInsight Storm	<a href="#">Write to Apache Hadoop HDFS from Apache Storm on HDInsight</a>

### Relational data

You can also source data from relational databases. Over a period of time, relational databases collect huge amounts of data which can provide key insights if processed through a big data pipeline. You can use the following tools to move such data into Data Lake Storage Gen2.

Here's a list of tools that you can use to ingest relational data.

TOOL	GUIDANCE
Azure Data Factory	<a href="#">Copy Activity in Azure Data Factory</a>

### Web server log data (upload using custom applications)

This type of dataset is specifically called out because analysis of web server log data is a common use case for big data applications and requires large volumes of log files to be uploaded to Data Lake Storage Gen2. You can use any of the following tools to write your own scripts or applications to upload such data.

Here's a list of tools that you can use to ingest Web server log data.

TOOL	GUIDANCE
Azure Data Factory	<a href="#">Copy Activity in Azure Data Factory</a>

TOOL	GUIDANCE
Azure CLI	<a href="#">Azure CLI</a>
Azure PowerShell	<a href="#">Azure PowerShell</a>

For uploading web server log data, and also for uploading other kinds of data (e.g. social sentiments data), it is a good approach to write your own custom scripts/applications because it gives you the flexibility to include your data uploading component as part of your larger big data application. In some cases this code may take the form of a script or simple command line utility. In other cases, the code may be used to integrate big data processing into a business application or solution.

### Data associated with Azure HDInsight clusters

Most HDInsight cluster types (Hadoop, HBase, Storm) support Data Lake Storage Gen2 as a data storage repository. HDInsight clusters access data from Azure Storage Blobs (WASB). For better performance, you can copy the data from WASB into a Data Lake Storage Gen2 account associated with the cluster. You can use the following tools to copy the data.

Here's a list of tools that you can use to ingest data associated with HDInsight clusters.

TOOL	GUIDANCE
Apache DistCp	<a href="#">Use DistCp to copy data between Azure Storage Blobs and Azure Data Lake Storage Gen2</a>
AzCopy tool	<a href="#">Transfer data with the AzCopy</a>
Azure Data Factory	<a href="#">Copy data to or from Azure Data Lake Storage Gen2 by using Azure Data Factory</a>

### Data stored in on-premises or IaaS Hadoop clusters

Large amounts of data may be stored in existing Hadoop clusters, locally on machines using HDFS. The Hadoop clusters may be in an on-premises deployment or may be within an IaaS cluster on Azure. There could be requirements to copy such data to Azure Data Lake Storage Gen2 for a one-off approach or in a recurring fashion. There are various options that you can use to achieve this. Below is a list of alternatives and the associated trade-offs.

APPROACH	DETAILS	ADVANTAGES	CONSIDERATIONS
Use Azure Data Factory (ADF) to copy data directly from Hadoop clusters to Azure Data Lake Storage Gen2	<a href="#">ADF supports HDFS as a data source</a>	ADF provides out-of-the-box support for HDFS and first class end-to-end management and monitoring	Requires Data Management Gateway to be deployed on-premises or in the IaaS cluster
Use Distcp to copy data from Hadoop to Azure Storage. Then copy data from Azure Storage to Data Lake Storage Gen2 using appropriate mechanism.	You can copy data from Azure Storage to Data Lake Storage Gen2 using: <ul style="list-style-type: none"> <li>• <a href="#">Azure Data Factory</a></li> <li>• <a href="#">AzCopy tool</a></li> <li>• <a href="#">Apache DistCp running on HDInsight clusters</a></li> </ul>	You can use open-source tools.	Multi-step process that involves multiple technologies

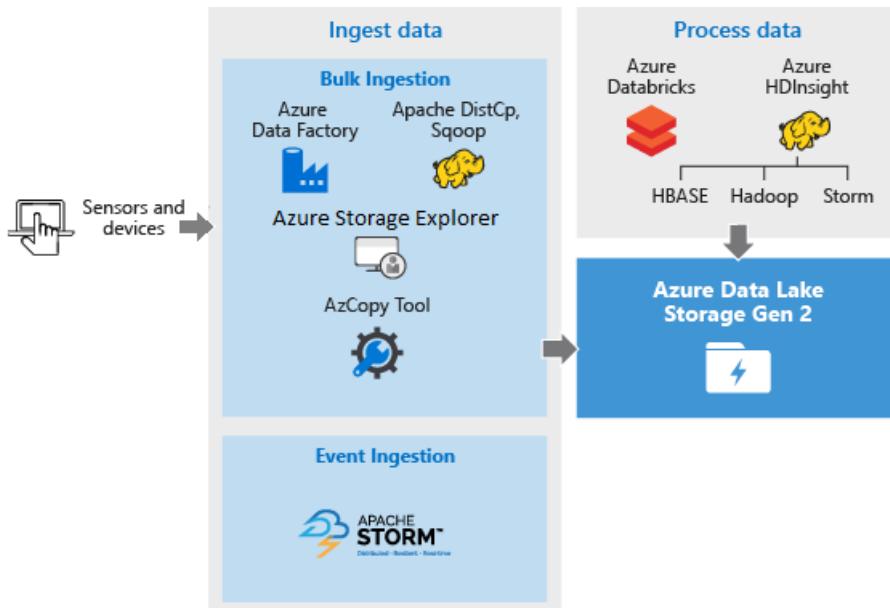
## Really large datasets

For uploading datasets that range in several terabytes, using the methods described above can sometimes be slow and costly. In such cases, you can use Azure ExpressRoute.

Azure ExpressRoute lets you create private connections between Azure data centers and infrastructure on your premises. This provides a reliable option for transferring large amounts of data. To learn more, see [Azure ExpressRoute documentation](#).

## Process the data

Once the data is available in Data Lake Storage Gen2 you can run analysis on that data using the supported big data applications.



Here's a list of tools that you can use to run data analysis jobs on data that is stored in Data Lake Storage Gen2.

TOOL	GUIDANCE
Azure HDInsight	<a href="#">Use Azure Data Lake Storage Gen2 with Azure HDInsight clusters</a>
Azure Databricks	<a href="#">Azure Data Lake Storage Gen2</a> <a href="#">Quickstart: Analyze data in Azure Data Lake Storage Gen2 by using Azure Databricks</a> <a href="#">Tutorial: Extract, transform, and load data by using Azure Databricks</a>

## Visualize the data

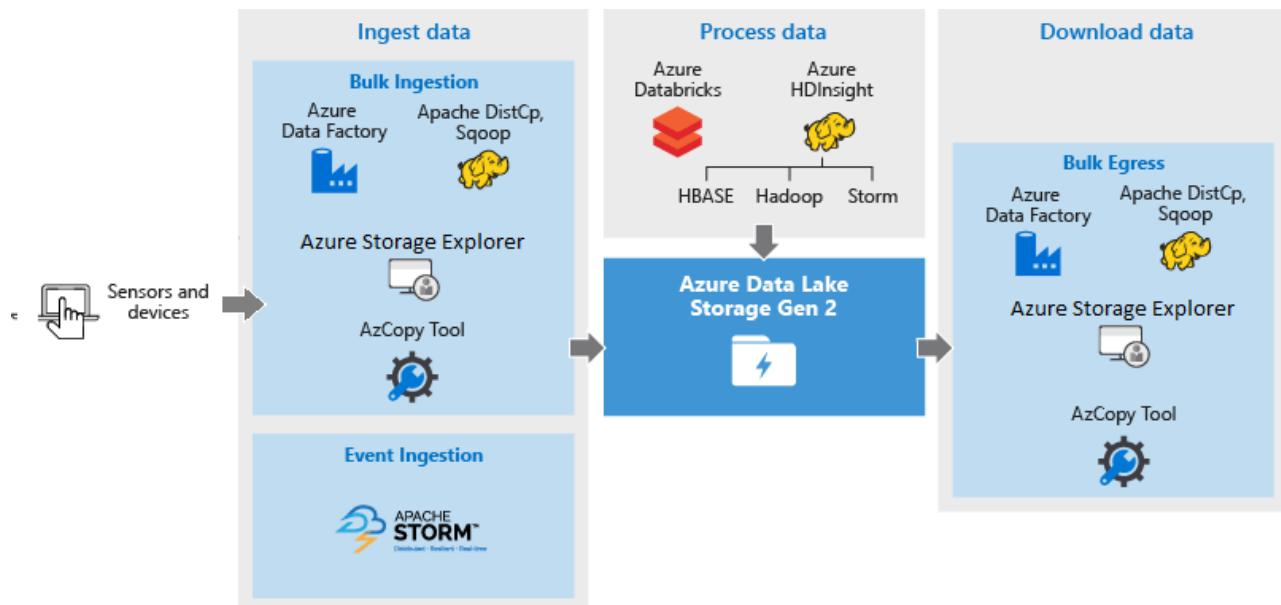
Use the Power BI connector to create visual representations of data stored in Data Lake Storage Gen2. See [Analyze data in Azure Data Lake Storage Gen2 by using Power BI](#).

## Download the data

You might also want to download or move data from Azure Data Lake Storage Gen2 for scenarios such as:

- Move data to other repositories to interface with your existing data processing pipelines. For example, you might want to move data from Data Lake Storage Gen2 to Azure SQL Database or on-premises SQL Server.

- Download data to your local computer for processing in IDE environments while building application prototypes.



Here's a list of tools that you can use to download data from Data Lake Storage Gen2.

TOOL	GUIDANCE
Azure Data Factory	<a href="#">Copy Activity in Azure Data Factory</a>
Apache DistCp	<a href="#">Use DistCp to copy data between Azure Storage Blobs and Azure Data Lake Storage Gen2</a>
Azure Storage Explorer	<a href="#">Use Azure Storage Explorer to manage directories, files, and ACLs in Azure Data Lake Storage Gen2</a>
AzCopy tool	<a href="#">Transfer data with AzCopy and Blob storage</a>

2 minutes to read

# Azure Data Lake Storage query acceleration (preview)

4/21/2020 • 4 minutes to read • [Edit Online](#)

Query acceleration (preview) is a new capability for Azure Data Lake Storage that 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.

## NOTE

The query acceleration feature is in public preview, and is available in the Canada Central and France Central regions. To review limitations, see the [Known issues](#) article. To enroll in the preview, see [this form](#).

## 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.

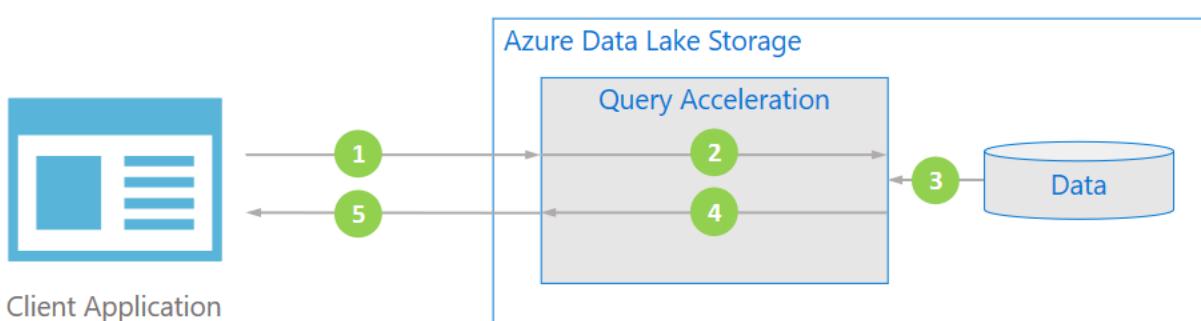
You can use SQL to specify the row filter predicates and column projections in a query acceleration request. A request processes only one file. Therefore, advanced relational features of SQL, such as joins and group by aggregates, aren't supported. Query acceleration supports CSV and JSON formatted data 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.

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

- [Query acceleration enrollment form](#)
- [Filter data by using Azure Data Lake Storage query acceleration](#)
- [Query acceleration SQL language reference \(preview\)](#)
- [Query acceleration REST API reference](#)

# The Azure Blob Filesystem driver (ABFS): A dedicated Azure Storage driver for Hadoop

1/14/2020 • 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@mynalytics.dfs.core.windows.net/tutorials/flightdelays/data
hdfs dfs -put flight_delays.csv
abfs://fileanalysis@mynalytics.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

3/31/2020 • 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. You have the option to connect with or without a Transport Layer Security (TLS), previously known as Secure Sockets Layer (SSL), connection. Use `abfss` to connect with a TLS connection.
- 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

2/11/2020 • 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.

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

- [Create a Storage account](#)

# Storage account overview

3/13/2020 • 14 minutes to read • [Edit Online](#)

An Azure storage account contains all of your Azure Storage data objects: blobs, files, queues, tables, and disks. The storage account provides a unique namespace for your Azure Storage data that is accessible from anywhere in the world over HTTP or HTTPS. Data in your Azure 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. Consider these differences before you create a storage account to determine the type of account that is best for your applications. The types of storage accounts are:

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

The following table describes the types of storage accounts and their capabilities:

STORAGE ACCOUNT TYPE	SUPPORTED SERVICES	SUPPORTED PERFORMANCE TIERS	SUPPORTED ACCESS TIERS	REPLICATION OPTIONS	DEPLOYMENT MODEL <sup>1</sup>	ENCRYPTION <sup>2</sup>
General-purpose V2	Blob, File, Queue, Table, Disk, and Data Lake Gen2 <sup>6</sup>	Standard, Premium <sup>5</sup>	Hot, Cool, Archive <sup>3</sup>	LRS, GRS, RA-GRS, ZRS, GZRS (preview), RA-GZRS (preview) <sup>4</sup>	Resource Manager	Encrypted
General-purpose V1	Blob, File, Queue, Table, and Disk	Standard, Premium <sup>5</sup>	N/A	LRS, GRS, RA-GRS	Resource Manager, Classic	Encrypted
BlockBlobStorage	Blob (block blobs and append blobs only)	Premium	N/A	LRS, ZRS <sup>4</sup>	Resource Manager	Encrypted

Storage account type	Supported services	Supported performance tiers	Supported access tiers	Replication options	Deployment model	Encryption
FileStorage	File only	Premium	N/A	LRS, ZRS <sup>4</sup>	Resource Manager	Encrypted
BlobStorage	Blob (block blobs and append blobs only)	Standard	Hot, Cool, Archive <sup>3</sup>	LRS, GRS, RA-GRS	Resource Manager	Encrypted

<sup>1</sup>Using the Azure Resource Manager deployment model is recommended. Storage accounts using the classic deployment model can still be created in some locations, and existing classic accounts continue to be supported. For more information, see [Azure Resource Manager vs. classic deployment: Understand deployment models and the state of your resources](#).

<sup>2</sup>All storage accounts are encrypted using Storage Service Encryption (SSE) for data at rest. For more information, see [Azure Storage Service Encryption for Data at Rest](#).

<sup>3</sup>Archive storage and blob-level tiering only support block blobs. The Archive tier is available at the level of an individual blob only, not at the storage account level. For more information, see [Azure Blob storage: Hot, Cool, and Archive storage tiers](#).

<sup>4</sup>Zone-redundant storage (ZRS) and geo-zone-redundant storage (GZRS/RA-GZRS) (preview) are available only for standard general-purpose V2, BlockBlobStorage, and FileStorage accounts in certain regions. For more information about Azure Storage redundancy options, see [Azure Storage redundancy](#).

<sup>5</sup>Premium performance for general-purpose v2 and general-purpose v1 accounts is available for disk and page blob only. Premium performance for block or append blobs are only available on BlockBlobStorage accounts. Premium performance for files are only available on FileStorage accounts.

<sup>6</sup>Azure Data Lake Storage Gen2 is a set of capabilities dedicated to big data analytics, built on Azure Blob storage. Data Lake Storage Gen2 is only supported on General-purpose V2 storage accounts with Hierarchical namespace enabled. For more information on Data Lake Storage Gen2, see [Introduction to Azure Data Lake Storage Gen2](#).

## General-purpose v2 accounts

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 deliver the lowest per-gigabyte capacity prices for Azure Storage, as well as industry-competitive transaction prices. General-purpose v2 storage accounts support these Azure Storage services:

- Blobs (all types: Block, Append, Page)
- Data Lake Gen2
- Files
- Disks
- Queues
- Tables

#### **NOTE**

Microsoft recommends using a general-purpose v2 storage account for most scenarios. You can easily upgrade a general-purpose v1 or Blob storage account to a general-purpose v2 account with no downtime and without the need to copy data.

For more information on upgrading to a general-purpose v2 account, see [Upgrade to a general-purpose v2 storage account](#).

General-purpose v2 storage accounts offer multiple access tiers for storing data based on your usage patterns. For more information, see [Access tiers for block blob data](#).

#### **General-purpose v1 accounts**

General-purpose v1 storage accounts provide access to all Azure Storage services, but may not have the latest features or the lowest per gigabyte pricing. General-purpose v1 storage accounts support these Azure Storage services:

- Blobs (all types)
- Files
- Disks
- Queues
- Tables

You should use general-purpose v2 accounts in most cases. You can use general-purpose v1 accounts for these scenarios:

- Your applications require the Azure classic deployment model. General-purpose v2 accounts and Blob storage accounts support only the Azure Resource Manager deployment model.
- Your applications are transaction-intensive or use significant geo-replication bandwidth, but don't require large capacity. In this case, general-purpose v1 may be the most economical choice.
- You use a version of the [Storage Services REST API](#) that is earlier than 2014-02-14 or a client library with a version lower than 4.x. You can't upgrade your application.

#### **BlockBlobStorage accounts**

A BlockBlobStorage account is a specialized storage account in the premium performance tier for storing unstructured object data as block blobs or append blobs. Compared with general-purpose v2 and BlobStorage accounts, BlockBlobStorage accounts provide low, consistent latency and higher transaction rates.

BlockBlobStorage accounts don't currently support tiering to hot, cool, or archive access tiers. This type of storage account does not support page blobs, tables, or queues.

#### **FileStorage accounts**

A FileStorage account is a specialized storage account used to store and create premium file shares. This storage account kind supports files but not block blobs, append blobs, page blobs, tables, or queues.

FileStorage accounts offer unique performance dedicated characteristics such as IOPS bursting. For more information on these characteristics, see the [File share storage tiers](#) section of the Files planning guide.

## **Naming storage accounts**

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.

## Performance tiers

Depending on the type of storage account you create, you can choose between standard and premium performance tiers.

### General-purpose storage accounts

General-purpose storage accounts may be configured for either of the following performance tiers:

- A standard performance tier for storing blobs, files, tables, queues, and Azure virtual machine disks. For more information about scalability targets for standard storage accounts, see [Scalability targets for standard storage accounts](#).
- A premium performance tier for storing unmanaged virtual machine disks. Microsoft recommends using managed disks with Azure virtual machines instead of unmanaged disks. For more information about scalability targets for the premium performance tier, see [Scalability targets for premium page blob storage accounts](#).

### BlockBlobStorage storage accounts

BlockBlobStorage storage accounts provide a premium performance tier for storing block blobs and append blobs. For more information, see [Scalability targets for premium block blob storage accounts](#).

### FileStorage storage accounts

FileStorage storage accounts provide a premium performance tier for Azure file shares. For more information, see [Azure Files scalability and performance targets](#).

## Access tiers for block blob data

Azure Storage provides different options for accessing block blob data based on usage patterns. Each access tier in Azure Storage is optimized for a particular pattern of data usage. By selecting the right access tier for your needs, you can store your block blob data in the most cost-effective manner.

The available access tiers are:

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

If there's a change in the usage pattern of your data, you can switch between these access tiers at any time. For more information about access tiers, see [Azure Blob storage: hot, cool, and archive access tiers](#).

#### IMPORTANT

Changing the access tier for an existing storage account or blob may result in additional charges. For more information, see the [Storage account billing section](#).

## Redundancy

Redundancy options for a storage account include:

- Locally redundant storage (LRS): A simple, low-cost redundancy strategy. Data is copied synchronously three times within the primary region.
- Zone-redundant storage (ZRS): Redundancy for scenarios requiring high availability. Data is copied synchronously across three Azure availability zones in the primary region.
- Geo-redundant storage (GRS): Cross-regional redundancy to protect against regional outages. Data is copied synchronously three times in the primary region, then copied asynchronously to the secondary region. For read access to data in the secondary region, enable read-access geo-redundant storage (RA-GRS).
- Geo-zone-redundant storage (GZRS) (preview): Redundancy for scenarios requiring both high availability and maximum durability. Data is copied synchronously across three Azure availability zones in the primary region, then copied asynchronously to the secondary region. For read access to data in the secondary region, enable read-access geo-zone-redundant storage (RA-GZRS).

For more information about redundancy options in Azure Storage, see [Azure Storage redundancy](#).

## Encryption

All data in your storage account is encrypted on the service side. For more information about encryption, see [Azure Storage Service Encryption for data at rest](#).

## Storage account endpoints

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

For example, if your general-purpose storage account is named *mystorageaccount*, then the default endpoints for that account are:

- Blob storage: `https://*mystorageaccount*.blob.core.windows.net`
- Table storage: `https://*mystorageaccount*.table.core.windows.net`
- Queue storage: `https://*mystorageaccount*.queue.core.windows.net`
- Azure Files: `https://*mystorageaccount*.file.core.windows.net`

### NOTE

Block blob and blob storage accounts expose only the Blob service endpoint.

Construct the URL for accessing an object in a storage account by appending the object's location in the storage account to the endpoint. For example, a blob address might have this format:  
`http://mystorageaccount.blob.core.windows.net/mycontainer/myblob`.

You can also configure your storage account to use a custom domain for blobs. For more information, see [Configure a custom domain name for your Azure Storage account](#).

## Control access to account data

By default, the data in your account is available only to you, the account owner. You have control over who may access your data and what permissions they have.

Every request made against your storage account must be authorized. At the level of the service, the request must include a valid *Authorization* header. Specifically, this header includes all of the information necessary for the service to validate the request before executing it.

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

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

#### NOTE

Authenticating users or applications using Azure AD credentials provides superior security and ease of use over other means of authorization. While you can continue to use Shared Key authorization with your applications, using Azure AD circumvents the need to store your account access key with your code. You can also continue to use shared access signatures (SAS) to grant fine-grained access to resources in your storage account, but Azure AD offers similar capabilities without the need to manage SAS tokens or worry about revoking a compromised SAS.

Microsoft recommends using Azure AD authorization for your Azure Storage blob and queue applications when possible.

## Copying data into a storage account

Microsoft provides utilities and libraries 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.

When you upgrade to a general-purpose v2 account from a general-purpose v1 or Blob storage account, your data is automatically migrated. Microsoft recommends this pathway for upgrading your account. However, if you decide to move data from a general-purpose v1 account to a Blob storage account, then you'll migrate your data manually, using the tools and libraries described below.

### AzCopy

AzCopy is a Windows command-line utility designed for high-performance copying of data to and from Azure Storage. You can use AzCopy to copy data into a Blob storage account from an existing general-purpose storage account, or to upload data from on-premises storage devices. For more information, see [Transfer data with the AzCopy Command-Line Utility](#).

### Data movement library

The Azure Storage data movement library for .NET is based on the core data movement framework that powers AzCopy. The library is designed for high-performance, reliable, and easy data transfer operations similar to AzCopy. You can use the data movement library to take advantage of AzCopy features natively. For more information, see [Azure Storage Data Movement Library for .NET](#)

### REST API or client library

You can create a custom application to migrate your data from a general-purpose v1 storage account into a

Blob storage account. Use one of the Azure client libraries or the Azure storage services REST API. Azure Storage provides rich client libraries for multiple languages and platforms like .NET, Java, C++, Node.js, PHP, Ruby, and Python. The client libraries offer advanced capabilities such as retry logic, logging, and parallel uploads. You can also develop directly against the REST API, which can be called by any language that makes HTTP/HTTPS requests.

For more information about the Azure Storage REST API, see [Azure Storage Services REST API Reference](#).

**IMPORTANT**

Blobs encrypted using client-side encryption store encryption-related metadata with the blob. If you copy a blob that is encrypted with client-side encryption, ensure that the copy operation preserves the blob metadata, and especially the encryption-related metadata. If you copy a blob without the encryption metadata, the blob content cannot be retrieved again. For more information regarding encryption-related metadata, see [Azure Storage Client-Side Encryption](#).

## Storage account billing

You're billed for Azure Storage 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.
- **Storage Capacity** refers to how much of your storage account allotment you're using to store data.
- **Replication** 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.

## Next steps

- [Create a storage account](#)
- [Create a block blob storage account](#)

# Best practices for using Azure Data Lake Storage Gen2

10/11/2019 • 10 minutes to read • [Edit Online](#)

In this article, you learn about best practices and considerations for working with Azure Data Lake Storage Gen2. This article provides information around security, performance, resiliency, and monitoring for Data Lake Storage Gen2. Before Data Lake Storage Gen2, working with truly big data in services like Azure HDInsight was complex. You had to shard data across multiple Blob storage accounts so that petabyte storage and optimal performance at that scale could be achieved. Data Lake Storage Gen2 supports individual file sizes as high as 5TB and most of the hard limits for performance have been removed. However, there are still some considerations that this article covers so that you can get the best performance with Data Lake Storage Gen2.

## Security considerations

Azure Data Lake Storage Gen2 offers POSIX access controls for Azure Active Directory (Azure AD) users, groups, and service principals. These access controls can be set to existing files and directories. The access controls can also be used to create default permissions that can be automatically applied to new files or directories. More details on Data Lake Storage Gen2 ACLs are available at [Access control in Azure Data Lake Storage Gen2](#).

### Use security groups versus individual users

When working with big data in Data Lake Storage Gen2, it is likely that a service principal is used to allow services such as Azure HDInsight to work with the data. However, there might be cases where individual users need access to the data as well. In all cases, strongly consider using Azure Active Directory [security groups](#) instead of assigning individual users to directories and files.

Once a security group is assigned permissions, adding or removing users from the group doesn't require any updates to Data Lake Storage Gen2. This also helps ensure you don't exceed the maximum number of access control entries per access control list (ACL). Currently, that number is 32, (including the four POSIX-style ACLs that are always associated with every file and directory): the owning user, the owning group, the mask, and other. Each directory can have two types of ACL, the access ACL and the default ACL, for a total of 64 access control entries. For more information about these ACLs, see [Access control in Azure Data Lake Storage Gen2](#).

### Security for groups

When you or your users need access to data in a storage account with hierarchical namespace enabled, it's best to use Azure Active Directory security groups. Some recommended groups to start with might be **ReadOnlyUsers**, **WriteAccessUsers**, and **FullAccessUsers** for the root of the container, and even separate ones for key subdirectories. If there are any other anticipated groups of users that might be added later, but have not been identified yet, you might consider creating dummy security groups that have access to certain folders. Using security group ensures that you can avoid long processing time when assigning new permissions to thousands of files.

### Security for service principals

Azure Active Directory service principals are typically used by services like Azure Databricks to access data in Data Lake Storage Gen2. For many customers, a single Azure Active Directory service principal might be adequate, and it can have full permissions at the root of the Data Lake Storage Gen2 container. Other customers might require multiple clusters with different service principals where one cluster has full access to the data, and another cluster with only read access.

### Enable the Data Lake Storage Gen2 firewall with Azure service access

Data Lake Storage Gen2 supports the option of turning on a firewall and limiting access only to Azure services, which is recommended to limit the vector of external attacks. Firewall can be enabled on a storage account in the Azure portal via the **Firewall > Enable Firewall (ON) > Allow access to Azure services** options.

To access your storage account from Azure Databricks, deploy Azure Databricks to your virtual network, and then add that virtual network to your firewall. See [Configure Azure Storage firewalls and virtual networks](#).

## Resiliency considerations

When architecting a system with Data Lake Storage Gen2 or any cloud service, you must consider your availability requirements and how to respond to potential interruptions in the service. An issue could be localized to the specific instance or even region-wide, so having a plan for both is important. Depending on the recovery time objective and the recovery point objective SLAs for your workload, you might choose a more or less aggressive strategy for high availability and disaster recovery.

### High availability and disaster recovery

High availability (HA) and disaster recovery (DR) can sometimes be combined together, although each has a slightly different strategy, especially when it comes to data. Data Lake Storage Gen2 already handles 3x replication under the hood to guard against localized hardware failures. Additionally, other replication options, such as ZRS or GZRS (preview), improve HA, while GRS & RA-GRS improve DR. When building a plan for HA, in the event of a service interruption the workload needs access to the latest data as quickly as possible by switching over to a separately replicated instance locally or in a new region.

In a DR strategy, to prepare for the unlikely event of a catastrophic failure of a region, it is also important to have data replicated to a different region using GRS or RA-GRS replication. You must also consider your requirements for edge cases such as data corruption where you may want to create periodic snapshots to fall back to. Depending on the importance and size of the data, consider rolling delta snapshots of 1-, 6-, and 24-hour periods, according to risk tolerances.

For data resiliency with Data Lake Storage Gen2, it is recommended to geo-replicate your data via GRS or RA-GRS that satisfies your HA/DR requirements. Additionally, you should consider ways for the application using Data Lake Storage Gen2 to automatically fail over to the secondary region through monitoring triggers or length of failed attempts, or at least send a notification to admins for manual intervention. Keep in mind that there is tradeoff of failing over versus waiting for a service to come back online.

### Use Distcp for data movement between two locations

Short for distributed copy, DistCp is a Linux command-line tool that comes with Hadoop and provides distributed data movement between two locations. The two locations can be Data Lake Storage Gen2, HDFS, or S3. This tool uses MapReduce jobs on a Hadoop cluster (for example, HDInsight) to scale out on all the nodes. Distcp is considered the fastest way to move big data without special network compression appliances. Distcp also provides an option to only update deltas between two locations, handles automatic retries, as well as dynamic scaling of compute. This approach is incredibly efficient when it comes to replicating things like Hive/Spark tables that can have many large files in a single directory and you only want to copy over the modified data. For these reasons, Distcp is the most recommended tool for copying data between big data stores.

Copy jobs can be triggered by Apache Oozie workflows using frequency or data triggers, as well as Linux cron jobs. For intensive replication jobs, it is recommended to spin up a separate HDInsight Hadoop cluster that can be tuned and scaled specifically for the copy jobs. This ensures that copy jobs do not interfere with critical jobs. If running replication on a wide enough frequency, the cluster can even be taken down between each job. If failing over to secondary region, make sure that another cluster is also spun up in the secondary region to replicate new data back to the primary Data Lake Storage Gen2 account once it comes back up. For examples of using Distcp, see [Use Distcp to copy data between Azure Storage Blobs and Data Lake Storage Gen2](#).

### Use Azure Data Factory to schedule copy jobs

Azure Data Factory can also be used to schedule copy jobs using a Copy Activity, and can even be set up on a

frequency via the Copy Wizard. Keep in mind that Azure Data Factory has a limit of cloud data movement units (DMUs), and eventually caps the throughput/compute for large data workloads. Additionally, Azure Data Factory currently does not offer delta updates between Data Lake Storage Gen2 accounts, so directories like Hive tables would require a complete copy to replicate. Refer to the [data factory article](#) for more information on copying with Data Factory.

## Monitoring considerations

Data Lake Storage Gen2 provides metrics in the Azure portal under the Data Lake Storage Gen2 account and in Azure Monitor. Availability of Data Lake Storage Gen2 is displayed in the Azure portal. To get the most up-to-date availability of a Data Lake Storage Gen2 account, you must run your own synthetic tests to validate availability. Other metrics such as total storage utilization, read/write requests, and ingress/egress are available to be leveraged by monitoring applications and can also trigger alerts when thresholds (for example, Average latency or # of errors per minute) are exceeded.

## Directory layout considerations

When landing data into a data lake, it's important to pre-plan the structure of the data so that security, partitioning, and processing can be utilized effectively. Many of the following recommendations are applicable for all big data workloads. Every workload has different requirements on how the data is consumed, but below are some common layouts to consider when working with IoT and batch scenarios.

### IoT structure

In IoT workloads, there can be a great deal of data being landed in the data store 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/
```

There's an important reason to put the date at the end of the directory structure. If you want to lock down certain regions or subject matters to users/groups, then you can easily do so with the POSIX permissions. Otherwise, if there was a need to restrict a certain security group to viewing just the UK data or certain planes, with the date structure in front a separate permission would be required for numerous directories under every hour directory. Additionally, having the date structure in front would exponentially increase the number of directories as time went on.

### Batch jobs structure

From a high-level, a commonly used approach in batch processing is to land data in 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 seen sometimes 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/producer). This structure helps with securing the data across your organization and better management of the data in your workloads. Furthermore, 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, 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** folder since 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 folders, and orchestration by something like 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.

# Azure Storage redundancy

4/16/2020 • 12 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.

When deciding which redundancy option is best for your scenario, consider the tradeoffs between lower costs and higher availability and durability. 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 location that is geographically distant to the primary region, to protect against regional disasters
- Whether your application requires read access to the replicated data in the secondary region if the primary region becomes unavailable for any reason

## Redundancy in the primary region

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

- **Locally redundant storage (LRS)** copies your data synchronously three times within a single physical location in the primary region. LRS is the least expensive replication option, but is not recommended for applications requiring high availability.
- **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.

### Locally-redundant storage

Locally redundant storage (LRS) replicates your data three times within a single physical location in the primary region. LRS provides at least 99.99999999% (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 \(preview\) \(GZRS\)](#).

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.

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](#).

### Zone-redundant storage

Zone-redundant storage (ZRS) replicates your Azure Storage data 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 Azure Storage data objects of at least 99.999999999% (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 re-pointing. 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 consistency, durability, and high availability. 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 following table shows which types of storage accounts support ZRS in which regions:

STORAGE ACCOUNT TYPE	SUPPORTED REGIONS	SUPPORTED SERVICES
General-purpose v2 <sup>1</sup>	Asia Southeast Australia East Europe North Europe West France Central Japan East South Africa North UK South US Central US East US East 2 US West 2	Block blobs Page blobs <sup>2</sup> File shares (standard) Tables Queues
BlockBlobStorage <sup>1</sup>	Europe West US East	Block blobs only
FileStorage	Europe West US East	Azure Files only

<sup>1</sup> The archive tier is not currently supported for ZRS accounts.

<sup>2</sup> Storage accounts that contain Azure managed disks for virtual machines always use LRS. Azure unmanaged disks should also use LRS. It is possible to create a storage account for Azure unmanaged disks that uses GRS, but it is not recommended due to potential issues with consistency over asynchronous geo-replication. Neither managed nor unmanaged disks support ZRS or GZRS. For more information on managed disks, see [Pricing for Azure managed disks](#).

For information about which regions support ZRS, see [Services support by region](#) in [What are Azure Availability Zones?](#).

## Redundancy in a secondary region

For applications requiring high availability, 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.
- **Geo-zone-redundant storage (GZRS) (preview)** 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.

The primary difference between GRS and GZRS is how data is replicated in the primary region. Within the secondary location, data is always replicated synchronously three times using LRS.

With GRS or GZRS, the data in the secondary location isn't available for read or write access unless there is a failover to the secondary region. For read access to the secondary location, 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 (preview). 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 account failover \(preview\)](#).

#### 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. Azure Storage 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 Azure Storage data objects of at least 99.9999999999999% (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.

### Geo-zone-redundant storage (preview)

Geo-zone-redundant storage (GZRS) (preview) 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 99.9999999999999% (16 9's) durability of objects over a given year.

Only general-purpose v2 storage accounts support GZRS and RA-GZRS. For more information about storage account types, see [Azure storage account overview](#). GZRS and RA-GZRS support block blobs, page blobs (except for VHD disks), files, tables, and queues.

GZRS and RA-GZRS are currently available for preview in the following regions:

- Asia Southeast
- Europe North
- Europe West
- Japan East
- UK South
- US East
- US East 2
- US Central
- US West 2

Microsoft continues to enable GZRS and RA-GZRS in additional Azure regions. Check the [Azure Service Updates](#) page regularly for information about supported regions.

For information on preview pricing, refer to GZRS preview pricing for [Blobs](#), [Files](#), [Queues](#), and [Tables](#).

#### **IMPORTANT**

Microsoft recommends against using preview features for production workloads.

## 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. However, that data is available to be read only if the customer or Microsoft initiates a failover from the primary to secondary region. When you enable read access to the secondary region, your data is available to be read if the primary region becomes unavailable. For read access to the secondary region, enable 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 always available for read access, so you can test your application to make sure that it will read from the secondary in the event of an outage. For more information about how to design your applications for high availability, see [Designing highly available applications using read-access geo-redundant storage](#).

When read access to the secondary is enabled, your data can be read from the secondary endpoint as well as from the primary endpoint for your storage account. 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 will not 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 are 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 following table shows how durable and available your data is in a given scenario, depending on which type of redundancy is in effect for your storage account:

SCENARIO	LRS	ZRS	GRS/RA-GRS	GZRS/RA-GZRS (PREVIEW)
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	Yes
A region-wide outage occurs	No	No	Yes	Yes
Read access to data in the secondary region if the primary region becomes unavailable	No	No	Yes (with RA-GRS)	Yes (with RA-GZRS)
Percent durability of objects over a given year <sup>1</sup>	at least 99.999999999% (11 9's)	at least 99.9999999999% (12 9's)	at least 99.99999999999999% (16 9's)	at least 99.99999999999999% (16 9's)
Supported storage account types <sup>2</sup>	GPv2, GPv1, BlockBlobStorage, BlobStorage, FileStorage	GPv2, BlockBlobStorage, FileStorage	GPv2, GPv1, BlobStorage	GPv2

SCENARIO	LRS	ZRS	GRS/RA-GRS	GZRS/RA-GZRS (PREVIEW)
Availability SLA for read requests <sup>1</sup>	At least 99.9% (99% for cool access tier)	At least 99.9% (99% for cool access tier)	At least 99.9% (99% for cool access tier) for GRS	At least 99.9% (99% for cool access tier) for GZRS
			At least 99.99% (99.9% for cool access tier) for RA-GRS	At least 99.99% (99.9% for cool access tier) for RA-GZRS
Availability SLA for write requests <sup>1</sup>	At least 99.9% (99% for cool access tier)	At least 99.9% (99% for cool access tier)	At least 99.9% (99% for cool access tier)	At least 99.9% (99% for cool access tier)

<sup>1</sup> For information about Azure Storage guarantees for durability and availability, see the [Azure Storage SLA](#).

<sup>2</sup> For information for storage account types, see [Storage account overview](#).

All data for all types of storage accounts and [all tiers \(including archive\)](#) are copied according to the redundancy option for the storage account. Objects including block blobs, append blobs, page blobs, queues, tables, and files are copied.

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.

## Data integrity

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

## See also

- [Check the Last Sync Time property for a storage account](#)
- [Change the redundancy option for a storage account](#)
- [Designing highly available applications using RA-GRS Storage](#)
- [Disaster recovery and account failover \(preview\)](#)

# Choose an Azure solution for data transfer

3/4/2020 • 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

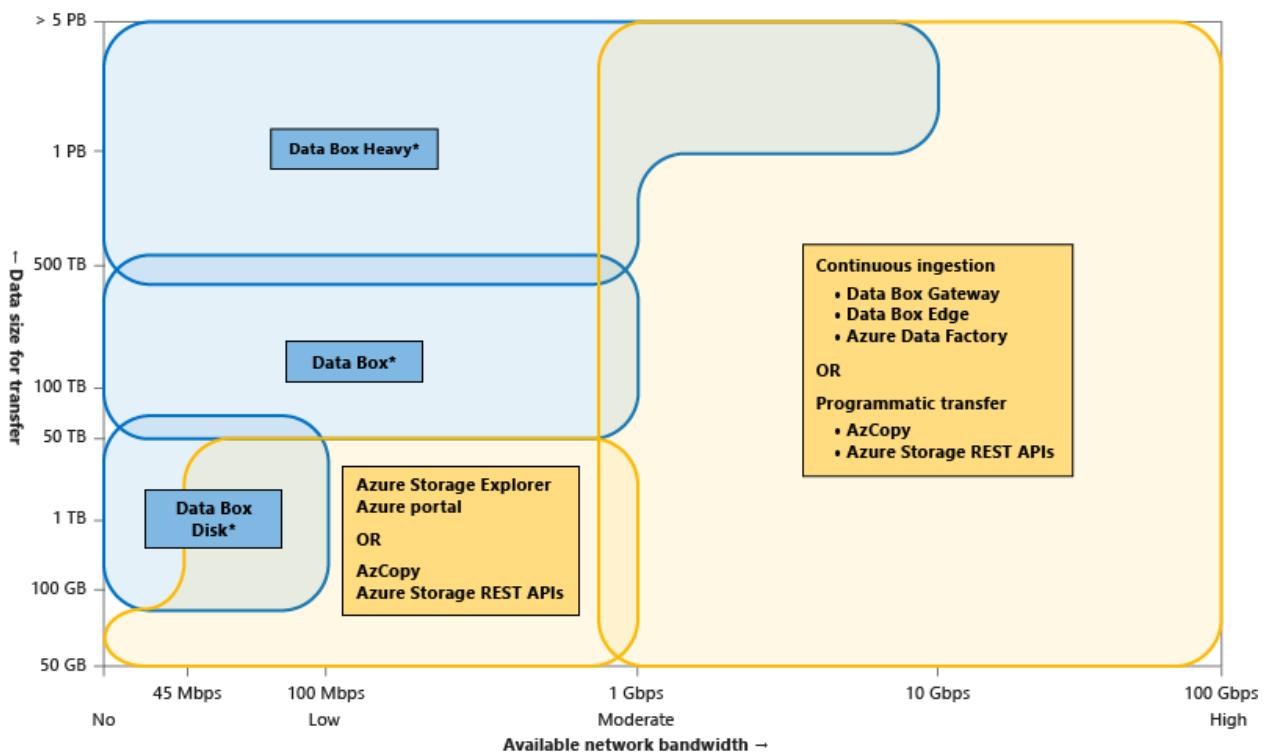
Data transfer can be offline or over the network connection. Choose your solution depending on your:

- **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 Data Box 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.](#)
- [Use Azure PowerShell with Azure Storage](#)
- [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 Data Box Edge for online transfers.](#)
- [Learn what is Azure Data Factory.](#)
- Use the REST APIs to transfer data
  - [In .NET](#)
  - [In Java](#)

# Managing Concurrency in Microsoft Azure Storage

12/23/2019 • 16 minutes to read • [Edit Online](#)

Modern Internet-based applications typically have multiple users viewing and updating data simultaneously. This requires application developers 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:

1. Optimistic concurrency – An application performing an update will verify if the data has changed since the application last read that data. For example, if two users viewing a wiki page make an update to the same page then the wiki platform must ensure that the second update does not overwrite the first update – and that both users understand whether their update was successful or not. This strategy is most often used in web applications.
2. 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 master/subordinate data replication scenario where only the master will perform updates the master will typically hold an exclusive lock for an extended period of time on the data to ensure no one else can update it.
3. Last writer wins – An approach that allows any update operations to proceed without verifying if any other application has updated the data since the application first read the data. This strategy (or lack of a formal strategy) is usually used where data is partitioned in such a way that there is no likelihood that multiple users will access the same data. It can also be useful where short-lived data streams are being processed.

This article provides an overview of how the Azure Storage platform simplifies development by providing first class support for all three of these concurrency strategies.

## Azure Storage simplifies cloud development

The Azure storage service supports all three strategies, although it is distinctive in its ability to provide full support for optimistic and pessimistic concurrency because it was designed to embrace a strong consistency model which guarantees that when the Storage service commits a data insert or update operation all further accesses to that data will see the latest update. Storage platforms that use an eventual consistency model have a lag between when a write is performed by one user and when the updated data can be seen by other users thus complicating development of client applications in order to prevent inconsistencies from affecting end users.

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. The Azure storage service uses snapshot isolation to allow read operations to happen concurrently with write operations within a single partition. Unlike other isolation levels, snapshot isolation guarantees that all reads see a consistent snapshot of the data even while updates are occurring – essentially by returning the last committed values while an update transaction is being processed.

## Managing concurrency in Blob storage

You can opt to use either optimistic or pessimistic concurrency models to manage access to blobs and containers in the Blob service. If you do not explicitly specify a strategy last writes wins is the default.

### Optimistic concurrency for blobs and containers

The Storage service assigns an identifier to every object stored. This identifier is updated every time an update operation is performed on an object. The identifier is returned to the client as part of an HTTP GET response using the ETag (entity tag) header that is defined within the HTTP protocol. A user performing an update on such an

object can send in the original ETag along with a conditional header to ensure that an update will only occur if a certain condition has been met – in this case the condition is an "If-Match" header, which requires the Storage Service to ensure the value of the ETag specified in the update request is the same as that stored in the Storage Service.

The outline of this process is as follows:

1. Retrieve a blob from the storage service, the response includes an HTTP ETag Header value that identifies the current version of the object in the storage service.
2. When you update the blob, include the ETag value you received in step 1 in the **If-Match** conditional header of the request you send to the service.
3. The service compares the ETag value in the request with the current ETag value of the blob.
4. If the current ETag value of the blob is a different version than the ETag in the **If-Match** conditional header in the request, the service returns a 412 error to the client. This indicates to the client that another process has updated the blob since the client retrieved it.
5. If the current ETag value of the blob is the same version as the ETag in the **If-Match** conditional header in the request, the service performs the requested operation and updates the current ETag value of the blob to show that it has created a new version.

The following C# snippet (using the Client Storage Library 4.2.0) shows a simple example of how to construct an **If-Match AccessCondition** based on the ETag value that is accessed from the properties of a blob that was previously either retrieved or inserted. It then uses the **AccessCondition** object when it updates the blob: the **AccessCondition** object adds the **If-Match** header to the request. If another process has updated the blob, the Blob service returns an HTTP 412 (Precondition Failed) status message. You can download the full sample here: [Managing Concurrency using Azure Storage](#).

```
// Retrieve the ETag from the newly created blob
// Etag is already populated as UploadText should cause a PUT Blob call
// to storage Blob service which returns the ETag in response.
string originalETag = blockBlob.Properties.ETag;

// This code simulates an update by a third party.
string helloText = "Blob updated by a third party.';

// No ETag provided so original blob is overwritten (thus generating a new ETag)
blockBlob.UploadText(helloText);
Console.WriteLine("Blob updated. Updated ETag = {0}",
 blockBlob.Properties.ETag);

// Now try to update the blob using the original ETag provided when the blob was created
try
{
 Console.WriteLine("Trying to update blob using original ETag to generate if-match access condition");
 blockBlob.UploadText(helloText,accessCondition:
 AccessCondition.GenerateIfMatchCondition(originalETag));
}
catch (StorageException ex)
{
 if (ex.RequestInformation.HttpStatusCode == (int)HttpStatusCode.PreconditionFailed)
 {
 Console.WriteLine("Precondition failure as expected. Blob's original ETag no longer matches");
 // TODO: client can decide on how it wants to handle the 3rd party updated content.
 }
 else
 throw;
}
```

Azure Storage also includes support for additional conditional headers such as **If-Modified-Since**, **If-Unmodified-Since** and **If-None-Match** as well as combinations thereof. For more information, see [Specifying](#)

## Conditional Headers for Blob Service Operations.

The following table summarizes the container operations that accept conditional headers such as **If-Match** in the request and that return an ETag value in the response.

OPERATION	RETURNS CONTAINER ETAG VALUE	ACCEPTS CONDITIONAL HEADERS
Create Container	Yes	No
Get Container Properties	Yes	No
Get Container Metadata	Yes	No
Set Container Metadata	Yes	Yes
Get Container ACL	Yes	No
Set Container ACL	Yes	Yes (*)
Delete Container	No	Yes
Lease Container	Yes	Yes
List Blobs	No	No

(\*) The permissions defined by SetContainerACL are cached and updates to these permissions take 30 seconds to propagate during which period updates are not guaranteed to be consistent.

The following table summarizes the blob operations that accept conditional headers such as **If-Match** in the request and that return an ETag value in the response.

OPERATION	RETURNS ETAG VALUE	ACCEPTS CONDITIONAL HEADERS
Put Blob	Yes	Yes
Get Blob	Yes	Yes
Get Blob Properties	Yes	Yes
Set Blob Properties	Yes	Yes
Get Blob Metadata	Yes	Yes
Set Blob Metadata	Yes	Yes
Lease Blob (*)	Yes	Yes
Snapshot Blob	Yes	Yes
Copy Blob	Yes	Yes (for source and destination blob)
Abort Copy Blob	No	No

OPERATION	RETURNS ETAG VALUE	ACCEPTS CONDITIONAL HEADERS
Delete Blob	No	Yes
Put Block	No	No
Put Block List	Yes	Yes
Get Block List	Yes	No
Put Page	Yes	Yes
Get Page Ranges	Yes	Yes

(\*) Lease Blob does not change the ETag on a blob.

### Pessimistic concurrency for blobs

To lock a blob for exclusive use, you can acquire a [lease](#) on it. When you acquire a lease, you specify for how long you need the lease: this can be for between 15 to 60 seconds or infinite, which amounts to an exclusive lock. You can renew a finite lease to extend it, and you can release any lease when you are finished with it. The Blob service automatically releases finite leases when they expire.

Leases enable different synchronization strategies to be supported, including exclusive write / shared read, exclusive write / exclusive read and shared write / exclusive read. Where a lease exists the storage service enforces exclusive writes (put, set and delete operations) however ensuring exclusivity for read operations requires the developer to ensure 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 C# snippet shows an example of acquiring an exclusive lease for 30 seconds on a blob, updating the content of the blob, and then releasing the lease. If there is already a valid lease on the blob when you try to acquire a new lease, the Blob service returns an "HTTP (409) Conflict" status result. The following snippet uses an **AccessCondition** object to encapsulate the lease information when it makes a request to update the blob in the storage service. You can download the full sample here: [Managing Concurrency using Azure Storage](#).

```
// Acquire lease for 15 seconds
string lease = blockBlob.AcquireLease(TimeSpan.FromSeconds(15), null);
Console.WriteLine("Blob lease acquired. Lease = {0}", lease);

// Update blob using lease. This operation will succeed
const string helloText = "Blob updated";
var accessCondition = AccessCondition.GenerateLeaseCondition(lease);
blockBlob.UploadText(helloText, accessCondition: accessCondition);
Console.WriteLine("Blob updated using an exclusive lease");

//Simulate third party update to blob without lease
try
{
 // Below operation will fail as no valid lease provided
 Console.WriteLine("Trying to update blob without valid lease");
 blockBlob.UploadText("Update without lease, will fail");
}
catch (StorageException ex)
{
 if (ex.RequestInformation.HttpStatusCode == (int)HttpStatusCode.PreconditionFailed)
 Console.WriteLine("Precondition failure as expected. Blob's lease does not match");
 else
 throw;
}
```

If you attempt a write operation on a leased blob without passing the lease ID, the request fails with a 412 error. Note that if the lease expires before calling the **UploadText** method but you still pass the lease ID, the request also fails with a 412 error. For more information about managing lease expiry times and lease IDs, see the [Lease Blob](#) REST documentation.

The following blob operations can use leases to manage pessimistic concurrency:

- Put Blob
- Get Blob
- Get Blob Properties
- Set Blob Properties
- Get Blob Metadata
- Set Blob Metadata
- Delete Blob
- Put Block
- Put Block List
- Get Block List
- Put Page
- Get Page Ranges
- Snapshot Blob - lease ID optional if a lease exists
- Copy Blob - lease ID required if a lease exists on the destination blob
- Abort Copy Blob - lease ID required if an infinite lease exists on the destination blob
- Lease Blob

### Pessimistic concurrency for containers

Leases on containers enable the same synchronization strategies to be supported as on blobs (exclusive write / shared read, exclusive write / exclusive read and shared write / exclusive read) however unlike blobs the storage service only enforces exclusivity 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 succeed on a leased container without including the lease ID in which case they are shared operations. If exclusivity of update (put or set) or read operations is required then developers should ensure all clients use a lease ID and that only one client at a time has a valid lease ID.

The following container operations can use leases to manage pessimistic concurrency:

- Delete Container
- Get Container Properties
- Get Container Metadata
- Set Container Metadata
- Get Container ACL
- Set Container ACL
- Lease Container

For more information, see:

- [Specifying Conditional Headers for Blob Service Operations](#)
- [Lease Container](#)
- [Lease Blob](#)

## Managing concurrency in Table storage

The Table service uses optimistic concurrency checks as the default behavior when you are working with entities,

unlike the Blob service where you must explicitly choose to perform optimistic concurrency checks. The other difference between the table and Blob services is that you can only manage the concurrency behavior of entities whereas with the Blob service you can manage the concurrency of both containers and blobs.

To use optimistic concurrency and to check if another process modified an entity since you retrieved it from the table storage service, you can use the ETag value you receive when the table service returns an entity. The outline of this process is as follows:

1. Retrieve an entity from the table storage service, the response includes an ETag value that identifies the current identifier associated with that entity in the storage service.
2. When you update the entity, include the ETag value you received in step 1 in the mandatory **If-Match** header of the request you send to the service.
3. The service compares the ETag value in the request with the current ETag value of the entity.
4. If the current ETag value of the entity is different than the ETag in the mandatory **If-Match** header in the request, the service returns a 412 error to the client. This indicates to the client that another process has updated the entity since the client retrieved it.
5. If the current ETag value of the entity is the same as the ETag in the mandatory **If-Match** header in the request or the **If-Match** header contains the wildcard character (\*), the service performs the requested operation and updates the current ETag value of the entity to show that it has been updated.

Note that unlike the Blob service, the table service requires the client to include an **If-Match** header in update requests. However, it is possible to force an unconditional update (last writer wins strategy) and bypass concurrency checks if the client sets the **If-Match** header to the wildcard character (\*) in the request.

The following C# snippet shows a customer entity that was previously either created or retrieved having their email address updated. The initial insert or retrieve operation stores the ETag value in the customer object, and because the sample uses the same object instance when it executes the replace operation, it automatically sends the ETag value back to the table service, enabling the service to check for concurrency violations. If another process has updated the entity in table storage, the service returns an HTTP 412 (Precondition Failed) status message. You can download the full sample here: [Managing Concurrency using Azure Storage](#).

```
try
{
 customer.Email = "updatedEmail@contoso.org";
 TableOperation replaceCustomer = TableOperation.Replace(customer);
 customerTable.Execute(replaceCustomer);
 Console.WriteLine("Replace operation succeeded.");
}
catch (StorageException ex)
{
 if (ex.RequestInformation.HttpStatusCode == 412)
 Console.WriteLine("Optimistic concurrency violation - entity has changed since it was retrieved.");
 else
 throw;
}
```

To explicitly disable the concurrency check, you should set the **ETag** property of the **employee** object to "\*" before you execute the replace operation.

```
customer.ETag = "*";
```

The following table summarizes how the table entity operations use ETag values:

OPERATION	RETURNS ETAG VALUE	REQUIRES IF-MATCH REQUEST HEADER
Query Entities	Yes	No
Insert Entity	Yes	No
Update Entity	Yes	Yes
Merge Entity	Yes	Yes
Delete Entity	No	Yes
Insert or Replace Entity	Yes	No
Insert or Merge Entity	Yes	No

Note that the **Insert or Replace Entity** and **Insert or Merge Entity** operations do *not* perform any concurrency checks because they do not send an ETag value to the table service.

In general developers using tables should rely on optimistic concurrency when developing scalable applications. If pessimistic locking is needed, one approach developers can take when accessing Tables is to assign a designated blob for each table and try to take a lease on the blob before operating on the table. This approach does require the application to ensure all data access paths obtain the lease prior to operating on the table. You should also note that the minimum lease time is 15 seconds which requires careful consideration for scalability.

For more information, see:

- [Operations on Entities](#)

## Managing Concurrency in the Queue Service

One scenario in which concurrency is a concern in the queueing service is where multiple clients are retrieving messages from a queue. When a message is retrieved from the queue, the response includes the message and a pop receipt value, which is required to delete the message. The message is not automatically deleted from the queue, but after it has been retrieved, it is not visible to other clients for the time interval specified by the visibilitytimeout parameter. The client that retrieves the message is expected to delete the message after it has been processed, and before the time specified by the TimeNextVisible element of the response, which is calculated based on the value of the visibilitytimeout parameter. The value of visibilitytimeout is added to the time at which the message is retrieved to determine the value of TimeNextVisible.

The queue service does not have support for either optimistic or pessimistic concurrency and for this reason clients processing messages retrieved from a queue should ensure messages are processed in an idempotent manner. A last writer wins strategy is used for update operations such as SetQueueServiceProperties, SetQueueMetaData, SetQueueACL and UpdateMessage.

For more information, see:

- [Queue Service REST API](#)
- [Get Messages](#)

## Managing concurrency in Azure Files

The file service can be accessed using two different protocol endpoints – SMB and REST. The REST service does not have support for either optimistic locking or pessimistic locking and all updates will follow a last writer wins strategy. SMB clients that mount file shares can leverage file system locking mechanisms to manage access to

shared files – including the ability to perform pessimistic locking. When an SMB client opens a file, it specifies both the file access and share mode. Setting a File Access option of "Write" or "Read/Write" along with a File Share mode of "None" will result in the file being locked by an SMB client until the file is closed. If REST operation is attempted on a file where an SMB client has the file locked the REST service will return status code 409 (Conflict) with error code SharingViolation.

When an SMB client opens a file for delete, it marks the file as pending delete until all other SMB client open handles on that file are closed. While a file is marked as pending delete, any REST operation on that file will return status code 409 (Conflict) with error code SMBDeletePending. Status code 404 (Not Found) is not returned since it is possible for the SMB client to remove the pending deletion flag prior to closing the file. In other words, status code 404 (Not Found) is only expected when the file has been removed. Note that while a file is in an SMB pending delete state, it will not be included in the List Files results. Also, note that the REST Delete File and REST Delete Directory operations are committed atomically and do not result in a pending delete state.

For more information, see:

- [Managing File Locks](#)

## Next steps

For the complete sample application referenced in this blog:

- [Managing Concurrency using Azure Storage - Sample Application](#)

For more information on Azure Storage see:

- [Microsoft Azure Storage Home Page](#)
- [Introduction to Azure Storage](#)
- Storage Getting Started for [Blob](#), [Table](#), [Queues](#), and [Files](#)
- Storage Architecture – [Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency](#)

# Authorizing access to data in Azure Storage

4/17/2020 • 2 minutes to read • [Edit Online](#)

Each time you access data in your storage account, your client makes a request over HTTP/HTTPS to Azure Storage. Every request to a secure resource must be authorized, so that the service ensures that the client has the permissions required to access the data.

The following table describes the options that Azure Storage offers for authorizing access to resources:

	SHARED KEY (STORAGE ACCOUNT KEY)	SHARED ACCESS SIGNATURE (SAS)	AZURE ACTIVE DIRECTORY (AZURE AD)	ON-PREMISES ACTIVE DIRECTORY DOMAIN SERVICES (PREVIEW)	ANONYMOUS PUBLIC READ ACCESS
Azure Blobs	<a href="#">Supported</a>	<a href="#">Supported</a>	<a href="#">Supported</a>	Not supported	<a href="#">Supported</a>
Azure Files (SMB)	<a href="#">Supported</a>	Not supported	Supported, only with AAD Domain Services	Supported, credentials must be synced to Azure AD	Not supported
Azure Files (REST)	<a href="#">Supported</a>	<a href="#">Supported</a>	Not supported	Not supported	Not supported
Azure Queues	<a href="#">Supported</a>	<a href="#">Supported</a>	<a href="#">Supported</a>	Not Supported	Not supported
Azure Tables	<a href="#">Supported</a>	<a href="#">Supported</a>	Not supported	Not supported	Not supported

Each authorization option is briefly described below:

- **Azure Active Directory (Azure AD) integration** for blobs, and queues. Azure AD provides role-based access control (RBAC) for control over a client's access to resources in a storage account. For more information regarding Azure AD integration for blobs and queues, see [Authorize access to Azure blobs and queues using Azure Active Directory](#).
- **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 RBAC for fine-grained control over a client's access to Azure Files resources in a storage account. For more information regarding Azure Files authentication using domain services, refer to the [overview](#).
- **On-premises Active Directory Domain Services (AD DS, or on-premises AD DS) authentication (preview)** 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 RBAC for share level access control and NTFS DACLs for directory/file level permission enforcement. For more information regarding Azure Files authentication using domain services, refer to the [overview](#).
- **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](#).
- **Shared access signatures** for blobs, files, queues, and tables. Shared access signatures (SAS) provide

limited delegated access to resources in a storage account. Adding constraints on the time interval for which the signature is valid or on permissions it grants provides flexibility in managing access. For more information, see [Using shared access signatures \(SAS\)](#).

- **Anonymous public read access** for containers and blobs. Authorization is not required. For more information, see [Manage anonymous read access to containers and blobs](#).

By default, all resources in Azure Storage are secured, and are available only to the account owner. Although you can use any of the authorization strategies outlined above to grant clients access to resources in your storage account, Microsoft recommends using Azure AD when possible for maximum security and ease of use.

## Next steps

- [Authorize access to Azure blobs and queues using Azure Active Directory](#)
- [Authorize with Shared Key](#)
- [Grant limited access to Azure Storage resources using shared access signatures \(SAS\)](#)

# Use the Azure Storage resource provider to access management resources

1/14/2020 • 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. RBAC provides fine-grained control over permissions to both types of resources. When you assign an RBAC 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 built-in RBAC role, see [Built-in roles for Azure resources](#).

Azure Storage supports using Azure AD to authorize requests against Blob and Queue storage. For information about RBAC roles for blob and queue data operations, see [Authorize access to blobs and queues using Active Directory](#).

## Assign management permissions with role-based access control (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 role-based access control (RBAC).

When you assign an RBAC 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 RBAC role to a security principal by using the [Azure portal](#), the [Azure CLI tools](#), [PowerShell](#), or the [Azure Storage resource provider REST API](#).

For more information about RBAC, see [What is role-based access control \(RBAC\) for Azure resources?](#) and [Classic subscription administrator roles](#), [Azure RBAC 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:

RBAC 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 assign 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 custom RBAC roles for access to management resources. For more information about custom roles, see [Custom roles for Azure resources](#).

## 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 role-based access control \(RBAC\) for Azure resources?](#)
- [Scalability targets for the Azure Storage resource provider](#)

# Access control in Azure Data Lake Storage Gen2

4/9/2020 • 15 minutes to read • [Edit Online](#)

Azure Data Lake Storage Gen2 implements an access control model that supports both Azure role-based access control (RBAC) and POSIX-like access control lists (ACLs). This article summarizes the basics of the access control model for Data Lake Storage Gen2.

## Role-based access control

RBAC uses role assignments to effectively 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) that is requesting access to Azure resources.

Typically, those Azure resources are constrained to top-level resources (For example: Azure Storage accounts). In the case of Azure Storage, and consequently Azure Data Lake Storage Gen2, this mechanism has been extended to the container (file system) resource.

To learn how to assign roles to security principals in the scope of your storage account, see [Grant access to Azure blob and queue data with RBAC in the Azure portal](#).

### NOTE

A guest user can't create a role assignment.

## The impact of role assignments on file and directory level access control lists

While using RBAC role assignments is a powerful mechanism to control access permissions, it is a very coarsely grained mechanism relative to ACLs. The smallest granularity for RBAC is at the container level and this will be evaluated at a higher priority than ACLs. Therefore, if you assign a role to a security principal in the scope of a container, that security principal has the authorization level associated with that role for ALL directories and files in that container, regardless of ACL assignments.

When a security principal is granted RBAC data permissions through a [built-in role](#), or through a custom role, these permissions are evaluated first upon authorization of a request. If the requested operation is authorized by the security principal's RBAC assignments then authorization is immediately resolved and no additional ACL checks are performed. Alternatively, if the security principal does not have an RBAC assignment, or the request's operation does not match the assigned permission, then ACL checks are performed to determine if the security principal is authorized to perform the requested operation.

### NOTE

If the security principal has been assigned the Storage Blob Data Owner built-in role assignment, then the security principal is considered a *super-user* and is granted full access to all mutating operations, including setting the owner of a directory or file as well as ACLs for directories and files for which they are not the owner. Super-user access is the only authorized manner to change the owner of a resource.

## Shared Key and Shared Access Signature (SAS) authentication

Azure Data Lake Storage Gen2 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 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.

## Access control lists on files and directories

You can associate a security principal with an access level for files and directories. These associations are captured in an *access control list (ACL)*. Each file and directory in your storage account has an access control list.

### NOTE

ACLs apply only to security principals in the same tenant.

If you assigned a role to a security principal at the storage account-level, you can use access control lists to grant that security principal elevated access to specific files and directories.

You can't use access control lists to provide a level of access that is lower than a level granted by a role assignment. For example, if you assign the [Storage Blob Data Contributor](#) role to a security principal, then you can't use access control lists to prevent that security principal from writing to a directory.

### Set file and directory level permissions by using access control lists

To set file and directory level permissions, see any of the following articles:

Azure Storage Explorer	<a href="#">Use Azure Storage Explorer to manage directories, files, and ACLs in Azure Data Lake Storage Gen2</a>
.NET	<a href="#">Use .NET to manage directories, files, and ACLs in Azure Data Lake Storage Gen2</a>
Java	<a href="#">Use Java to manage directories, files, and ACLs in Azure Data Lake Storage Gen2</a>
Python	<a href="#">Use Python to manage directories, files, and ACLs in Azure Data Lake Storage Gen2</a>
PowerShell	<a href="#">Use PowerShell to manage directories, files, and ACLs in Azure Data Lake Storage Gen2</a>
Azure CLI	<a href="#">Use Azure CLI to manage directories, files, and ACLs in Azure Data Lake Storage Gen2</a>
REST API	Path - Update

### 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 access control lists

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 a container object 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 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 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 permissions

The following table lists some common scenarios to help you understand which permissions are needed to perform certain operations on a storage account.

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.

## Access check algorithm

The following pseudocode represents the access check algorithm for storage accounts.

```

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)
 for entry in entries:
 if (user_is_member_of_group(user, entry.identity)) :
 member_count += 1
 perms |= entry.permissions
 if (member_count>0) :
 return ((desired_perms & perms & mask) == desired_perms)

 # 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.

#### NOTE

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. 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 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, regardless of what the default ACL indicates.

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)
```

## Common questions about ACLs in Data Lake Storage Gen2

### **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 ACLs. 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 simply need to add or remove them from the appropriate Azure AD security group. Keep in mind that ACLs are not inherited and so reapplying ACLs requires updating the ACL on every file and subdirectory.

### **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**.

### Does Data Lake Storage Gen2 support inheritance of ACLs?

Azure RBAC assignments do inherit. Assignments flow from subscription, resource group, and storage account resources down to the container resource.

ACLs do not inherit. However, default ACLs can be used to set ACLs for child subdirectories and files created under the parent directory.

### 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

- [Overview of Azure Data Lake Storage Gen2](#)

# Azure Storage encryption for data at rest

4/16/2020 • 2 minutes to read • [Edit Online](#)

Azure Storage automatically encrypts 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.

## About Azure Storage 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](#).

## About encryption key management

Data in a new storage account is encrypted with 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. If you choose to manage encryption with your own keys, you have two options:

- You can specify a *customer-managed key* with Azure Key Vault to use for encrypting and decrypting data in Blob storage and in Azure Files.<sup>1,2</sup> For more information about customer-managed keys, see [Use customer-managed keys with Azure Key Vault to manage 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 \(preview\)](#).

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

	MICROSOFT-MANAGED KEYS	CUSTOMER-MANAGED KEYS	CUSTOMER-PROVIDED KEYS
Encryption/decryption operations	Azure	Azure	Azure
Azure Storage services supported	All	Blob storage, Azure Files <sup>1,2</sup>	Blob storage
Key storage	Microsoft key store	Azure Key Vault	Customer's own key store
Key rotation responsibility	Microsoft	Customer	Customer
Key control	Microsoft	Customer	Customer

<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](#).

## Next steps

- [What is Azure Key Vault?](#)
- [Configure customer-managed keys for Azure Storage encryption from the Azure portal](#)
- [Configure customer-managed keys for Azure Storage encryption from PowerShell](#)
- [Configure customer-managed keys for Azure Storage encryption from Azure CLI](#)

# Use customer-managed keys with Azure Key Vault to manage Azure Storage encryption

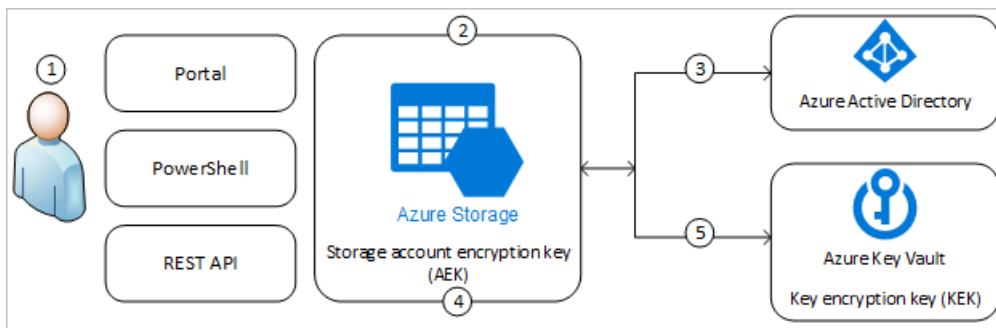
4/16/2020 • 5 minutes to read • [Edit Online](#)

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 Azure Key Vault to store your customer-managed keys. You can either create your own keys and store them in a key vault, or you can use the Azure Key Vault APIs to generate keys. The storage account and the key vault must be in the same region and in the same Azure Active Directory (Azure AD) tenant, but they can be in different subscriptions. For more information about Azure Key Vault, see [What is Azure Key Vault?](#).

## About customer-managed keys

The following diagram shows how Azure Storage uses Azure Active Directory and Azure Key Vault 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 the managed identity that's 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 that's associated with the storage account to authenticate access to Azure Key Vault via Azure Active Directory.
4. Azure Storage wraps the account encryption key with the customer 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.

## Create an account that supports customer-managed keys for queues and tables

Data stored in the Queue and Table services 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 at the time that you create the storage account to be included in this protection.

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 the Blob and File services 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

Customer-managed keys can be enabled only on existing storage accounts. The key vault must be provisioned with access policies that grant key permissions to the managed identity that is associated with the storage account. The managed identity is available only after the storage account is created.

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. Enabling customer-managed keys does not impact performance, and takes effect immediately.

When you modify the key being used for Azure Storage encryption by enabling or disabling customer-managed keys, updating the key version, or specifying a different key, then the encryption of the root key changes, but the data in your Azure Storage account does not need to be re-encrypted.

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.

To learn how to use customer-managed keys with Azure Key Vault for Azure Storage encryption, see one of these articles:

- [Configure customer-managed keys with Key Vault for Azure Storage encryption from the Azure portal](#)
- [Configure customer-managed keys with Key Vault for Azure Storage encryption from PowerShell](#)
- [Configure customer-managed keys with Key Vault for Azure Storage encryption from Azure CLI](#)

## IMPORTANT

Customer-managed keys rely on managed identities for Azure resources, a feature of Azure AD. Managed identities do not currently support cross-directory 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 directory 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](#).

## Store customer-managed keys in Azure Key Vault

To enable customer-managed keys on a storage account, you must use an Azure Key Vault to store your keys. You must enable both the **Soft Delete** and **Do Not Purge** properties on the key vault.

Only 2048-bit RSA and RSA-HSM keys are supported with Azure Storage encryption. For more information about keys, see [Key Vault keys](#) in [About Azure Key Vault keys, secrets and certificates](#).

## Rotate customer-managed keys

You can rotate a customer-managed key in Azure Key Vault according to your compliance policies. When the key is rotated, you must update the storage account to use the new key version URI. To learn how to update the storage account to use a new version of the key in the Azure portal, see the section titled **Update the key version** in [Configure customer-managed keys for Azure Storage by using the Azure portal](#).

Rotating the key does not trigger re-encryption of data in the storage account. There is no further action required from the user.

## 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

- [Configure customer-managed keys with Key Vault for Azure Storage encryption from the Azure portal](#)
- [Configure customer-managed keys with Key Vault for Azure Storage encryption from PowerShell](#)
- [Configure customer-managed keys with Key Vault for Azure Storage encryption from Azure CLI](#)
- [Azure Storage encryption for data at rest](#)

# Provide an encryption key on a request to Blob storage (preview)

3/15/2020 • 2 minutes to read • [Edit Online](#)

Clients making requests against Azure Blob storage have the option to provide an encryption key on a per-request basis (preview). 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 does not 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 is not 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 can have its own encryption 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. Must be AES256.

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 re-upload 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.

## Next steps

- [Specify a customer-provided key on a request to Blob storage with .NET](#)
- [Azure Storage encryption for data at rest](#)

# Configure advanced threat protection for Azure Storage

4/16/2020 • 3 minutes to read • [Edit Online](#)

Advanced threat protection for Azure 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 [Azure Security Center](#), 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 diagnostic logs of read, write, and delete requests to Blob Storage for threat detection. To investigate the alerts from advanced threat protection, 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

Advanced threat protection for Azure Storage is currently available only for [Blob Storage](#). Account types that support advanced threat protection include general-purpose v2, block blob, and Blob storage accounts. Advanced threat protection is available in all public clouds and US government clouds, but not in other sovereign or Azure government cloud regions.

For pricing details, including a free 30 day trial, see the [Azure Security Center pricing page](#).

## Set up advanced threat protection

You can configure advanced threat protection in any of several ways, described in the following sections.

- [Portal](#)
- [Azure Security Center](#)
- [Template](#)
- [Azure Policy](#)
- [REST API](#)
- [PowerShell](#)

1. Launch the [Azure portal](#).
2. Navigate to your Azure Storage account. Under **Settings**, select **Advanced security**.
3. Select the **Settings** link on the advanced security configuration page.
4. Set **Advanced security** to **ON**.
5. Click **Save** to save the new or updated policy.

The screenshot shows the 'Advanced security' section of the Azure Storage security settings. The left sidebar lists various security options: Access keys, Geo-replication, CORS, Configuration, Encryption, Shared access signature, Firewalls and virtual networks, Private endpoint connection..., Advanced security (which is selected and highlighted with a red box), Static website, Properties, Locks, Export template, Blob service, Blobs, Custom domain, and Soft delete. At the top right, there are 'Settings' and 'Refresh' buttons, and a link to 'Visit Security Center'. Below the sidebar, three main sections are displayed: 'Recommendations' (0 findings), 'Security alerts' (0 findings), and 'Settings'. The 'Settings' section shows that 'Advanced Threat Protection' is enabled (indicated by a green checkmark) and set to 'Enabled' (with a 'Settings' link). It also shows the 'Security tier' is 'Security Center Standard' and approximately 0 transactions per day are being analyzed. A 'Recommendations (preview)' section follows, stating that Azure Security Center monitors for vulnerabilities and recommends actions. It shows a list of three recommendations, all of which have been completed (indicated by checkmarks). A message indicates 'No recommendations to display'. Another message states 'There are no security recommendations for this resource'. A blue button at the bottom right says 'View all recommendations in Security Center'.

## 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	3887487-000a-411f-8d0-16037603774
Storage account	mystorageaccount
Storage type	Blob
Container	mycontainer
Application	myTestApplication
IP address	134.80.46.99
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 Azure Security Center's [Security alerts tile](#). Clicking on a specific alert provides details and actions for investigating the current threat and addressing future threats.

Anonymous access	
mystorageaccount	
<a href="#">Learn more</a>	
<h3>General Information</h3>	
DESCRIPTION	Someone has anonymously accessed your Azure Storage account "mystorageaccount"
DETECTION TIME	Friday, August 17, 2018, 10:50:00 AM
SEVERITY	<span style="color: orange;">⚠</span> Medium
STATE	Active
ATTACHED RESOURCE	mystorageresource
SUBSCRIPTION	[REDACTED]
DETECTED BY	 Microsoft
ENVIRONMENT	Azure
RESOURCE TYPE	 Storage
STORAGE ACCOUNT	mystorageaccount
STORAGE TYPE	Blob
CONTAINER	mycontainer
LOCATION	Washington, United States
USER AGENT	testUserAgentHeader
IP ADDRESS	[REDACTED]
DATA CENTER	East US

## 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 the **Storage** section in [Threat protection for data services in Azure Security Center](#).

## Next steps

- Learn more about [Logs in Azure Storage accounts](#)
- Learn more about [Azure Security Center](#)

# Use private endpoints for Azure Storage

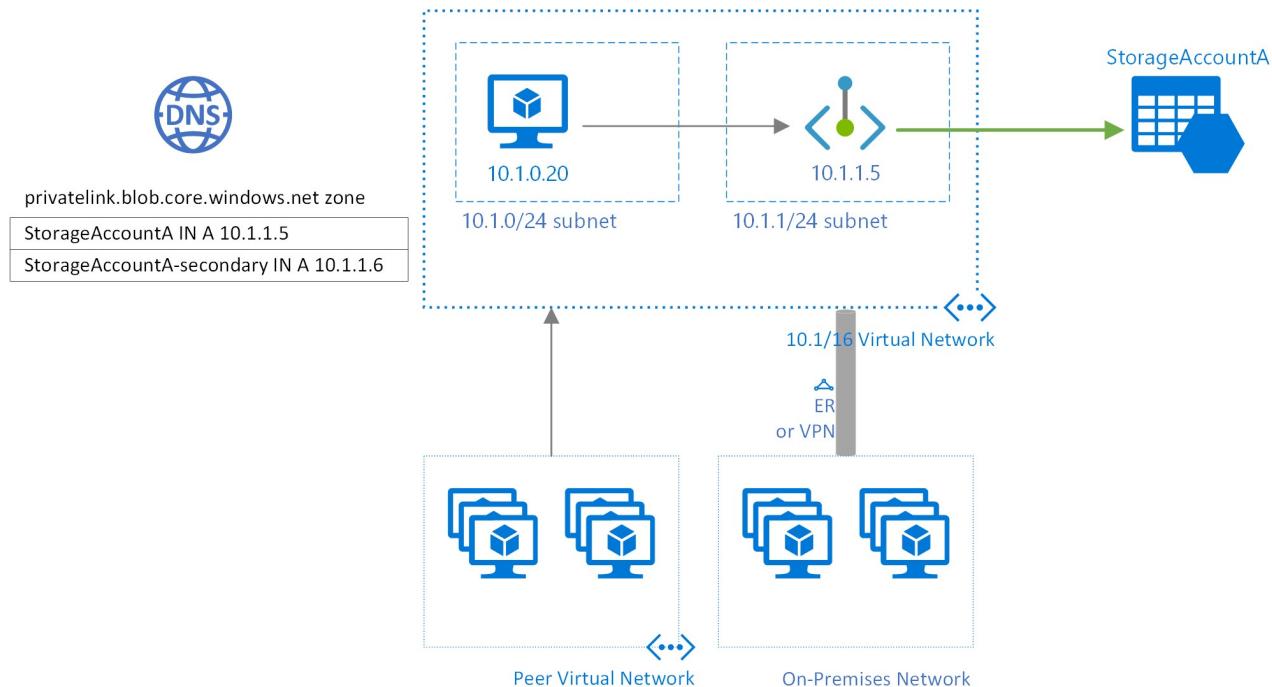
3/13/2020 • 7 minutes to read • [Edit Online](#)

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 an IP address from the VNet address space for your 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.

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.

#### Private endpoints for Azure Storage

When creating the 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 service in a storage account that you need to access, namely [Blobs](#), [Data Lake Storage Gen2](#), [Files](#), [Queues](#), [Tables](#), or [Static Websites](#).

#### TIP

Create a separate private endpoint for the secondary instance of the storage service for better read performance on RA-GRS accounts.

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](#).

For more detailed information on creating a private endpoint for your storage account, refer to the following articles:

- [Connect privately to a storage account from the Storage Account experience in the Azure portal](#)
- [Create a private endpoint using the Private Link Center in the Azure portal](#)
- [Create a private endpoint using Azure CLI](#)
- [Create a private endpoint using Azure PowerShell](#)

#### Connecting to private endpoints

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.

We create a [private DNS zone](#) attached to the VNet with the necessary updates for the private endpoints, by default. 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
StorageAccountA.blob.core.windows.net	CNAME	StorageAccountA.privatelink.blob.core.windows.net
StorageAccountA.privatelink.blob.core.windows.net	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
StorageAccountA.blob.core.windows.net	CNAME	StorageAccountA.privatelink.blob.core.windows.net
StorageAccountA.privatelink.blob.core.windows.net	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 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 are:

STORAGE SERVICE	ZONE NAME
Blob service	privatelink.blob.core.windows.net
Data Lake Storage Gen2	privatelink.dfs.core.windows.net
File service	privatelink.file.core.windows.net
Queue service	privatelink.queue.core.windows.net
Table service	privatelink.table.core.windows.net
Static Websites	privatelink.web.core.windows.net

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.

### **Copy Blob support**

If the storage account is protected by a firewall and the account is accessed through private endpoints, then that account cannot serve as the source of a [Copy Blob](#) operation.

### **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 instance, 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 applied to the private endpoint. 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.

## Next steps

- [Configure Azure Storage firewalls and virtual networks](#)
- [Security recommendations for Blob storage](#)

# Multi-protocol access on Azure Data Lake Storage

2/28/2020 • 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

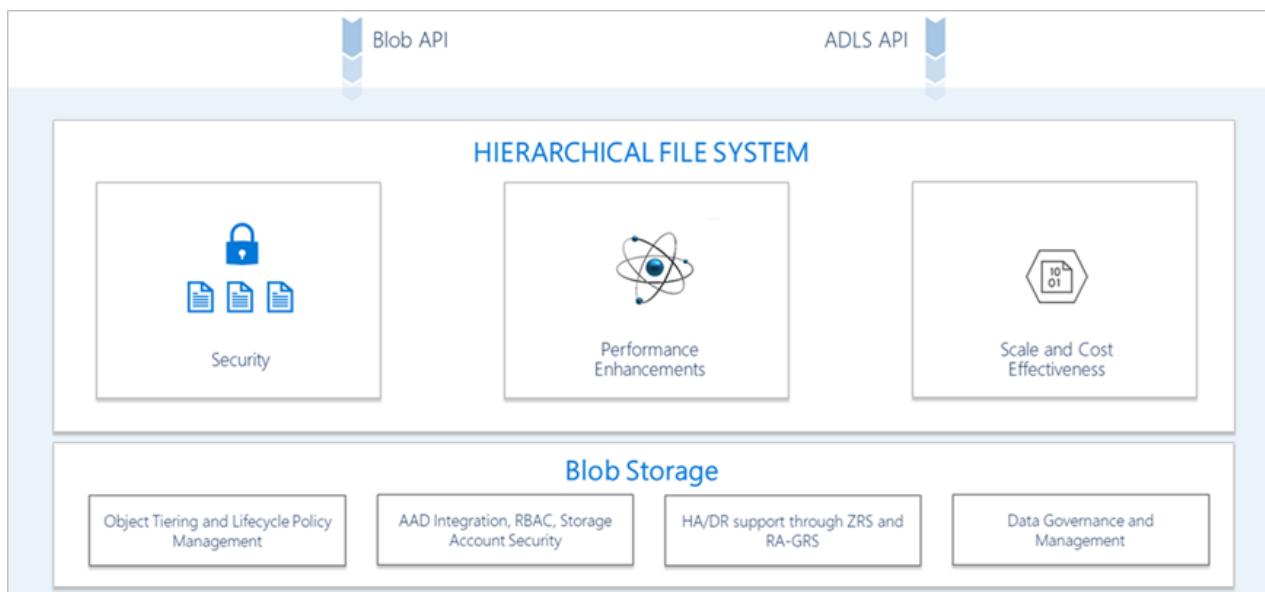
Multi-protocol access on Data Lake Storage is generally available and is available in all regions. Some Azure services or Blob storage features enabled by multi-protocol access remain in preview. These articles summarize the current support for Blob storage features and Azure service integrations.

[Blob Storage features available in Azure Data Lake Storage Gen2](#)

[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 features available in Azure Data Lake Storage Gen2](#)
- [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 features available in Azure Data Lake Storage Gen2

4/3/2020 • 2 minutes to read • [Edit Online](#)

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 features.

This table lists the Blob storage features 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.

## Supported Blob storage features

### NOTE

Support level refers only to how the feature is supported with Data Lake Storage Gen2.

BLOB STORAGE FEATURE	SUPPORT LEVEL	RELATED ARTICLES
Hot access tier	Generally available	<a href="#">Azure Blob storage: hot, cool, and archive access tiers</a>
Cool access tier	Generally available	<a href="#">Azure Blob storage: hot, cool, and archive access tiers</a>
Events	Generally available	<a href="#">Reacting to Blob storage events</a>
Metrics (Classic)	Generally available	<a href="#">Azure Storage analytics metrics (Classic)</a>
Metrics in Azure Monitor	Generally available	<a href="#">Azure Storage metrics in Azure Monitor</a>
Blob storage PowerShell commands	Generally available	<a href="#">Quickstart: Upload, download, and list blobs with PowerShell</a>
Blob storage Azure CLI commands	Generally available	<a href="#">Quickstart: Create, download, and list blobs with Azure CLI</a>
Blob storage APIs	Generally available	<a href="#">Quickstart: Azure Blob storage client library v12 for .NET</a> <a href="#">Quickstart: Manage blobs with Java v12 SDK</a> <a href="#">Quickstart: Manage blobs with Python v12 SDK</a> <a href="#">Quickstart: Manage blobs with JavaScript v12 SDK in Node.js</a>
Archive Access Tier	Preview	<a href="#">Azure Blob storage: hot, cool, and archive access tiers</a>
Lifecycle management policies	Preview	<a href="#">Manage the Azure Blob storage lifecycle</a>

BLOB STORAGE FEATURE	SUPPORT LEVEL	RELATED ARTICLES
Diagnostic logs	Generally available	<a href="#">Azure Storage analytics logging</a>
Change feed	Not yet supported	<a href="#">Change feed support in Azure Blob storage</a>
Account failover	Not yet supported	<a href="#">Disaster recovery and account failover</a>
Blob container ACL	Not yet supported	<a href="#">Set Container ACL</a>
Custom domains	Not yet supported	<a href="#">Map a custom domain to an Azure Blob storage endpoint</a>
Immutable storage	Not yet supported	<a href="#">Store business-critical blob data with immutable storage</a>
Snapshots	Not yet supported	<a href="#">Create and manage a blob snapshot in .NET</a>
Soft Delete	Not yet supported	<a href="#">Soft delete for Azure Storage blobs</a>
Static websites	Not yet supported	<a href="#">Static website hosting in Azure Storage</a>
Logging in Azure Monitor	Not yet supported	Not yet available
Premium block blobs	Not yet supported	<a href="#">Create a BlockBlobStorage account</a>

## See also

- [Known issues with Azure Data Lake Storage Gen2](#)
- [Azure services that support Azure Data Lake Storage Gen2](#)
- [Open source platforms that support Azure Data Lake Storage Gen2](#)
- [Multi-protocol access on Azure Data Lake Storage](#)

# Azure services that support Azure Data Lake Storage Gen2

4/7/2020 • 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	RELATED ARTICLES
Azure Data Factory	Generally available	<a href="#">Load data into Azure Data Lake Storage Gen2 with Azure Data Factory</a>
Azure Databricks	Generally available	<a href="#">Use with Azure Databricks</a> <a href="#">Quickstart: Analyze data in Azure Data Lake Storage Gen2 by using Azure Databricks</a> <a href="#">Tutorial: Extract, transform, and load data by using Azure Databricks</a> <a href="#">Tutorial: Access Data Lake Storage Gen2 data with Azure Databricks using Spark</a>
Azure Event Hub	Generally available	<a href="#">Capture events through Azure Event Hubs in Azure Blob Storage or Azure Data Lake Storage</a>
Azure Event Grid	Generally available	<a href="#">Tutorial: Implement the data lake capture pattern to update a Databricks Delta table</a>
Azure Logic Apps	Generally available	<a href="#">Overview - What is Azure Logic Apps?</a>
Azure Machine Learning	Generally available	<a href="#">Access data in Azure storage services</a>
Azure Stream Analytics	Generally available	<a href="#">Quickstart: Create a Stream Analytics job by using the Azure portal</a> <a href="#">Egress to Azure Data Lake Gen2</a>
Data Box	Generally available	<a href="#">Use Azure Data Box to migrate data from an on-premises HDFS store to Azure Storage</a>

AZURE SERVICE	SUPPORT LEVEL	RELATED ARTICLES
HDInsight	Generally available	<a href="#">Use Azure Data Lake Storage Gen2 with Azure HDInsight clusters</a> <a href="#">Using the HDFS CLI with Data Lake Storage Gen2</a> <a href="#">Tutorial: Extract, transform, and load data by using Apache Hive on Azure HDInsight</a>
IoT Hub	Generally available	<a href="#">Use IoT Hub message routing to send device-to-cloud messages to different endpoints</a>
Power BI	Generally available	<a href="#">Analyze data in Data Lake Storage Gen2 using Power BI</a>
SQL Data Warehouse	Generally available	<a href="#">Use with Azure SQL Data Warehouse</a>
SQL Server Integration Services (SSIS)	Generally available	<a href="#">Azure Storage connection manager</a>
Azure Cognitive Search	Preview	<a href="#">Index and search Azure Data Lake Storage Gen2 documents (preview)</a>
Azure Data Explorer	Preview	<a href="#">Query data in Azure Data Lake using Azure Data Explorer</a>
Azure Content Delivery Network	Not yet supported	<a href="#">Index and search Azure Data Lake Storage Gen2 documents (preview)</a>

## See also

- [Known issues with Azure Data Lake Storage Gen2](#)
- [Blob storage features available in Azure Data Lake Storage Gen2](#)
- [Open source platforms that support Azure Data Lake Storage Gen2](#)
- [Multi-protocol access on Azure Data Lake Storage](#)

# Open source platforms that support Azure Data Lake Storage Gen2

2/28/2020 • 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 features available in Azure Data Lake Storage Gen2](#)
- [Azure services that support Azure Data Lake Storage Gen2](#)
- [Multi-protocol access on Azure Data Lake Storage](#)

# Monitor, diagnose, and troubleshoot Microsoft Azure Storage

4/17/2020 • 55 minutes to read • [Edit Online](#)

## 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.

### NOTE

Azure Files does not support logging at this time.

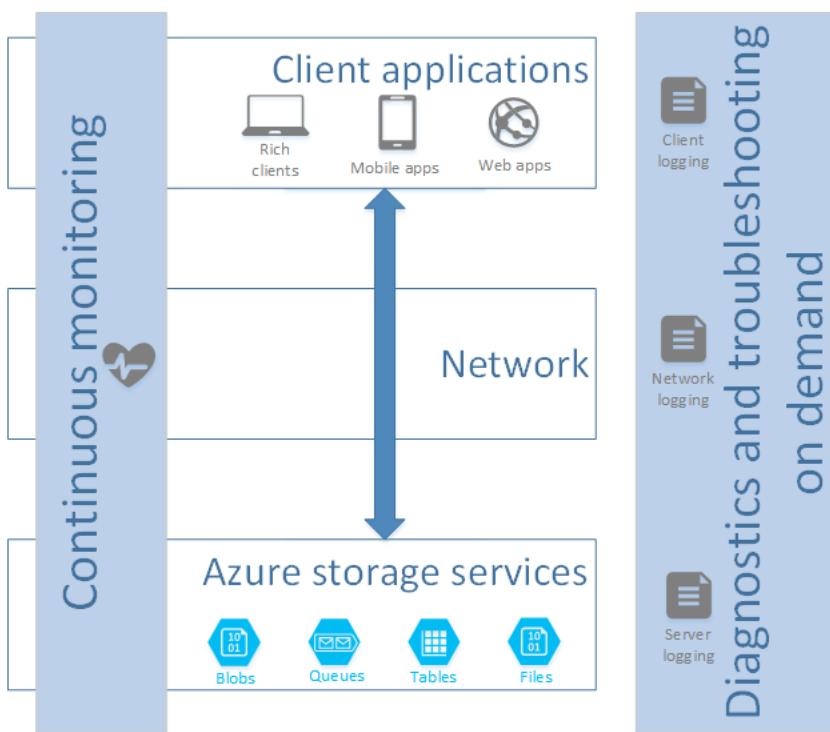
For a hands-on guide to end-to-end troubleshooting in Azure Storage applications, see [End-to-End Troubleshooting using Azure Storage Metrics and Logging, AzCopy, and Message Analyzer](#).

- [Introduction](#)
  - [How this guide is organized](#)
- [Monitoring your storage service](#)
  - [Monitoring service health](#)
  - [Monitoring capacity](#)
  - [Monitoring availability](#)
  - [Monitoring performance](#)
- [Diagnosing storage issues](#)
  - [Service health issues](#)
  - [Performance issues](#)
  - [Diagnosing errors](#)
  - [Storage emulator issues](#)
  - [Storage logging tools](#)
  - [Using network logging tools](#)
- [End-to-end tracing](#)
  - [Correlating log data](#)
  - [Client request ID](#)
  - [Server request ID](#)
  - [Timestamps](#)
- [Troubleshooting guidance](#)
  - [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](#)
  - [Metrics show an increase in PercentThrottlingError](#)
  - [Metrics show an increase in PercentTimeoutError](#)
  - [Metrics show an increase in PercentNetworkError](#)
  - [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
- Troubleshooting VHDs on Windows virtual machines
- Troubleshooting VHDs on Linux virtual machines
- Troubleshooting Azure Files issues with Windows
- Troubleshooting Azure Files issues with Linux
- Appendices
  - Appendix 1: Using Fiddler to capture HTTP and HTTPS traffic
  - Appendix 2: Using Wireshark to capture network traffic
  - Appendix 3: Using Microsoft Message Analyzer to capture network traffic
  - Appendix 4: Using Excel to view metrics and log data
  - Appendix 5: Monitoring with Application Insights for Azure DevOps

## Introduction

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.

### 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, Fiddler for analyzing HTTP/HTTPS messages, and Microsoft Message Analyzer for correlating log data.

## 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.

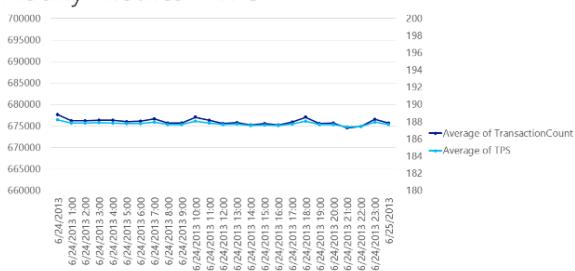
In the Azure portal, you can view metrics such as availability, total requests, and average latency numbers for a storage account. A notification rule has also been set up to alert an administrator if availability drops below a certain level. From viewing this data, one possible area for investigation 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:

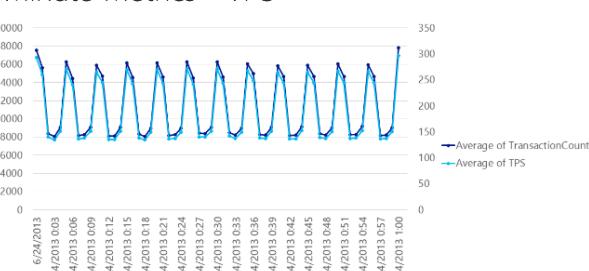
- Establishing some baseline metrics for application that will enable you to compare current data and identify any significant changes in the behavior of Azure storage and your application. The values of your baseline metrics will, in many cases, be application specific and you should establish them 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.

Hourly metrics - TPS



Minute Metrics - TPS



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](#).

While the [Azure portal](#) collects health information from inside the Azure datacenters (inside-out monitoring), you could also consider adopting an outside-in approach to generate synthetic transactions that periodically access your Azure-hosted web application from multiple locations. The services offered by [Dynatrace](#) and Application Insights for Azure DevOps are examples of this approach. 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](#)".
- Microsoft Message Analyzer is a tool from Microsoft that supersedes Netmon and that in addition to capturing network packet data, helps you to view and analyze the log data captured from other tools. For more information, see "[Appendix 3: Using Microsoft Message Analyzer 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. Message Analyzer can operate at both HTTP and TCP layers.

## 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.

If the Storage Client Library throws a **StorageException** in the client, the **RequestInformation** property contains a **RequestResult** object that includes a **ServiceRequestId** property. You can also access a **RequestResult** object from an **OperationContext** instance.

The code sample below demonstrates how to set a custom **ClientRequestId** value by attaching an **OperationContext** object the request to the storage service. It also shows how to retrieve the **ServerRequestId** value from the response message.

```
//Parse the connection string for the storage account.
const string ConnectionString = "DefaultEndpointsProtocol=https;AccountName=account-name;AccountKey=account-key";
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(ConnectionString);
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

// Create an Operation Context that includes custom ClientRequestId string based on constants defined within the application along
// with a Guid.
OperationContext oc = new OperationContext();
oc.ClientRequestId = String.Format("{0} {1} {2} {3}", HOSTNAME, APPNAME, USERID, Guid.NewGuid().ToString());

try
{
 CloudBlobContainer container = blobClient.GetContainerReference("democontainer");
 ICloudBlob blob = container.GetBlobReferenceFromServer("testImage.jpg", null, null, oc);
 var downloadToPath = string.Format("./{0}", blob.Name);
 using (var fs = File.OpenWrite(downloadToPath))
 {
 blob.DownloadToStream(fs, null, null, oc);
 Console.WriteLine("\t Blob downloaded to file: {0}", downloadToPath);
 }
}
catch (StorageException storageException)
{
 Console.WriteLine("Storage exception {0} occurred", storageException.Message);
 // Multiple results may exist due to client side retry logic - each retried operation will have a unique ServiceRequestId
 foreach (var result in oc.RequestResults)
 {
 Console.WriteLine("HttpStatus: {0}, ServiceRequestId {1}", result.HttpStatusCode, result.ServiceRequestId);
 }
}
```

#### 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

[You are experiencing unexpected reboots of Virtual Machines that have a large number of attached VHDs]

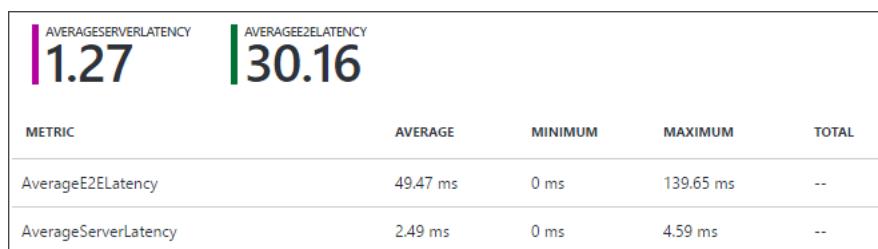
Your issue arises from using the storage emulator for development or test

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 acknowledgement 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.

```
var storageAccount = CloudStorageAccount.Parse(connStr);
ServicePoint tableServicePoint = ServicePointManager.FindServicePoint(storageAccount.TableEndpoint);
tableServicePoint.UseNagleAlgorithm = false;
ServicePoint queueServicePoint = ServicePointManager.FindServicePoint(storageAccount.QueueEndpoint);
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 or Microsoft Message Analyzer.

For more information about using Wireshark to troubleshoot network issues, see "[Appendix 2: Using Wireshark to capture network traffic](#)."

For more information about using Microsoft Message Analyzer to troubleshoot network issues, see "[Appendix 3: Using Microsoft Message Analyzer 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 or Microsoft Message Analyzer to investigate network issues.

For more information about using Wireshark to troubleshoot network issues, see "[Appendix 2: Using Wireshark to capture network traffic](#)."

For more information about using Microsoft Message Analyzer to troubleshoot network issues, see "[Appendix 3: Using Microsoft Message Analyzer 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 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 acknowledgement, 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 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, Microsoft Message Analyzer, 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://domemaildist.blob.core.windows.netazureimblblobcontainer/blobCreatedViaSAS.txt?sv=2014-02-14&amp;sr=c&amp;si=mypolicy&amp;sig=OFnd4Rd7z01fh%2BmcR6zbudIH2F51km%2FyhNYZEmJNQ%3D&amp;api-version=2014-02-14">https://domemaildist.blob.core.windows.netazureimblblobcontainer/blobCreatedViaSAS.txt?sv=2014-02-14&amp;sr=c&amp;si=mypolicy&amp;sig=OFnd4Rd7z01fh%2BmcR6zbudIH2F51km%2FyhNYZEmJNQ%3D&amp;api-version=2014-02-14</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 = 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..

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 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-...	<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.

REQUEST ID	OPERATION
e2d06d78...	CreateIfNotExists method to create the blob container. Note that this operation includes a HEAD request that checks for the existence of the container. The HEAD returns a 404 message but continues.
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 <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 = .
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> .

REQUEST ID	OPERATION TEXT
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 <a href="https://domemaildist.blob.core.windows.net/azuremmblobcontainer">https://domemaildist.blob.core.windows.net/azuremmblobcontainer</a> .
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 = .
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, 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?sv=2014-02-14&amp;sr=c&amp;si=mypolicy&amp;sig=XXXXX&amp;api-version=2014-02-14">https://domemaildist.blob.core.windows.net/azureimblobcontainer/blobCreate?sv=2014-02-14&amp;sr=c&amp;si=mypolicy&amp;sig=XXXXX&amp;api-version=2014-02-14</a>
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:

```
CloudBlobClient client = new CloudBlobClient(blobEndpoint, new StorageCredentials(accountName, accountKey));
// Set the service properties.
ServiceProperties sp = client.GetServiceProperties();
sp.DefaultServiceVersion = "2013-08-15";
CorsRule cr = new CorsRule();
cr.AllowedHeaders.Add("*");
cr.AllowedMethods = CorsHttpMethods.Get | CorsHttpMethods.Put;
cr.AllowedOrigins.Add("http://www.contoso.com");
cr.ExposedHeaders.Add("x-ms-*");
cr.MaxAgeInSeconds = 5;
sp.Cors.CorsRules.Clear();
sp.Cors.CorsRules.Add(cr);
client.SetServiceProperties(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 acknowledgement 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 acknowledgements 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 **CreateIfNotExists** 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, and Microsoft Message Analyzer 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](#)."

For more information about using Microsoft Message Analyzer, see "[Appendix 3: Using Microsoft Message Analyzer 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

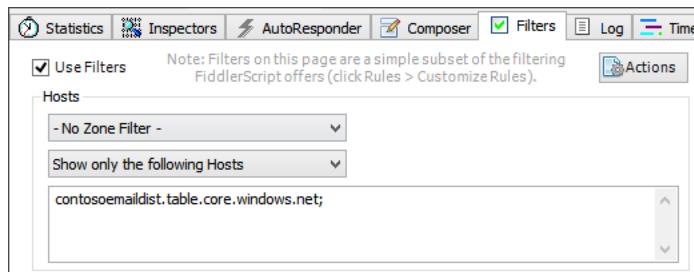
Fiddler can decode HTTPS traffic; you should read the Fiddler documentation carefully to understand how it does this, and to understand the security implications.

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:

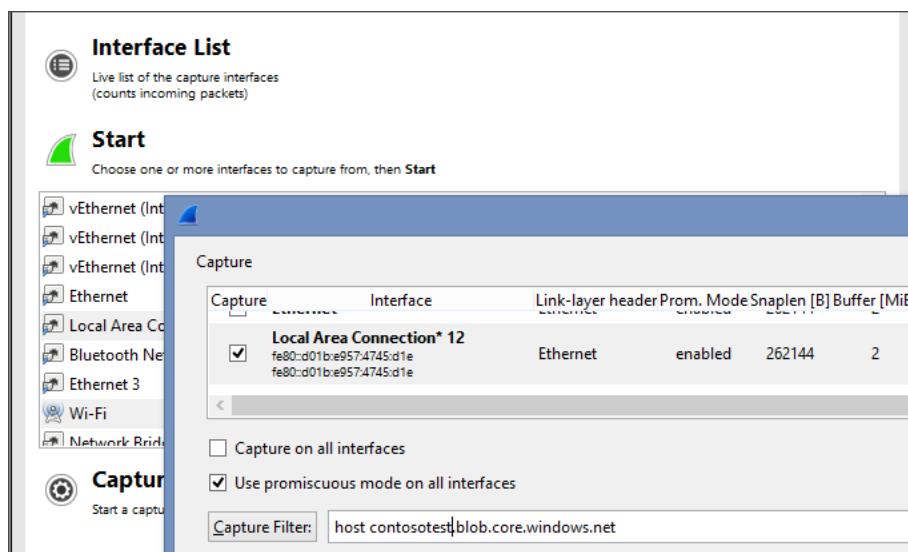


## Appendix 2: Using Wireshark to capture network traffic

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.

- Launch Wireshark on your local machine.
- In the **Start** section, select the local network interface or interfaces that are connected to the internet.
- Click **Capture Options**.
- 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](#).

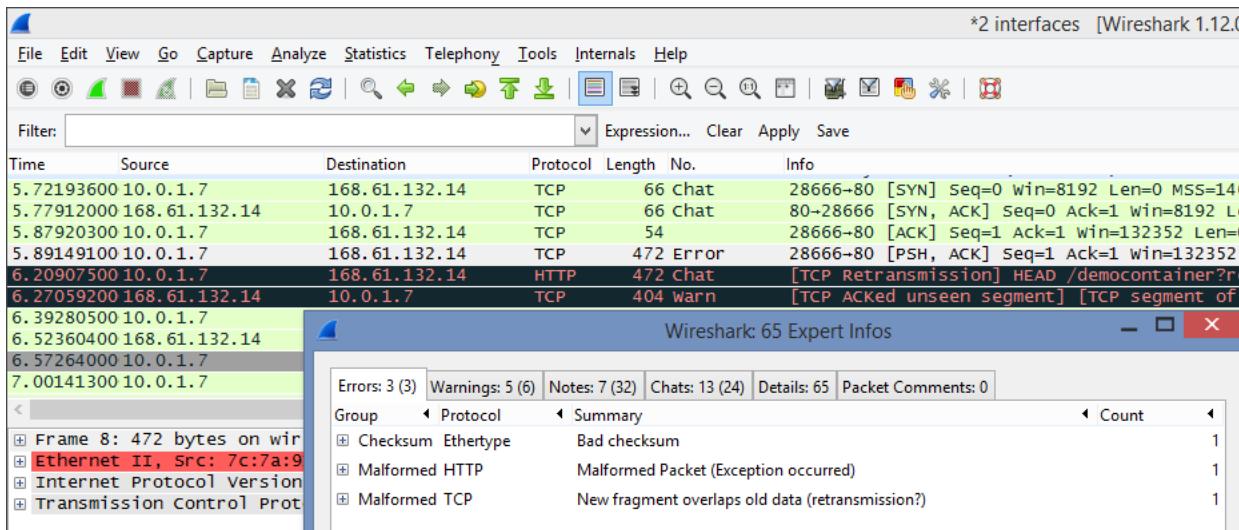


- Click **Start**. Wireshark will now capture all the packets sent to or from the table service endpoint as you use your client application on your local machine.

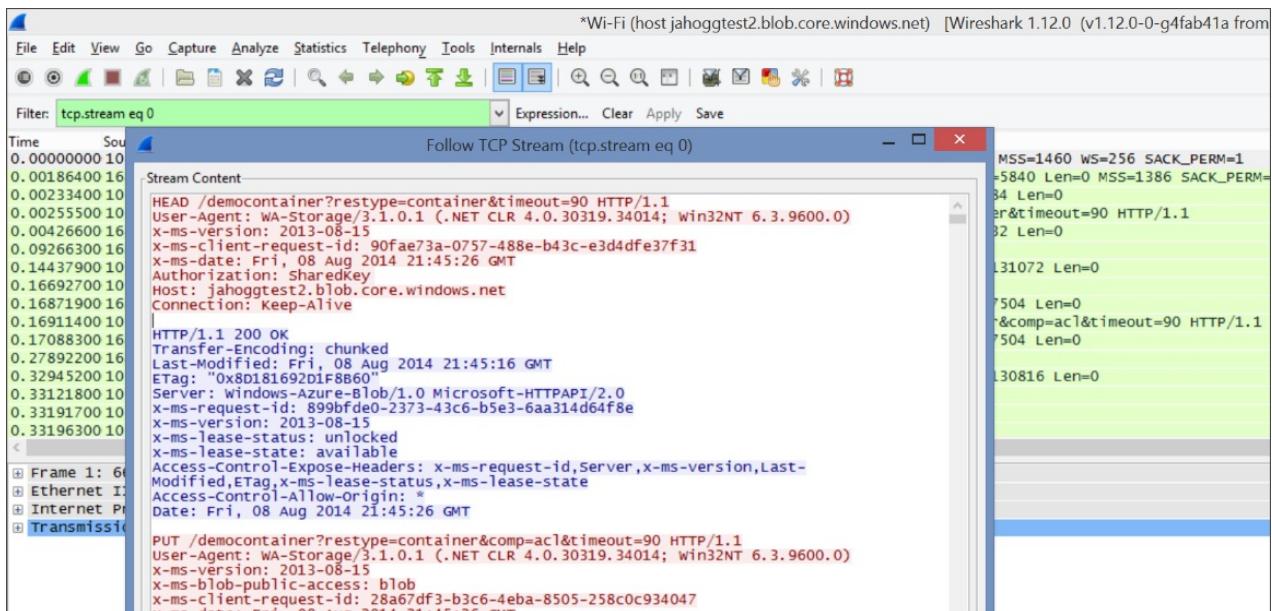
6. When you have finished, on the main menu click **Capture** and then **Stop**.

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.



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 3: Using Microsoft Message Analyzer to capture network traffic

You can use Microsoft Message Analyzer to capture HTTP and HTTPS traffic in a similar way to Fiddler, and capture network traffic in a similar way to Wireshark.

#### Configure a web tracing session using Microsoft Message Analyzer

To configure a web tracing session for HTTP and HTTPS traffic using Microsoft Message Analyzer, run the Microsoft Message Analyzer application and then on the **File** menu, click **Capture/Trace**. In the list of available trace scenarios, select **Web Proxy**. Then in the **Trace Scenario Configuration** panel, in the **HostnameFilter** textbox, add the names of your storage endpoints (you can look up these names in the [Azure portal](#)). For example, if the name of your Azure storage account is **contosodata**, you should add the following to the **HostnameFilter** textbox:

```
contosodata.blob.core.windows.net contosodata.table.core.windows.net contosodata.queue.core.windows.net
```

#### NOTE

A space character separates the hostnames.

When you are ready to start collecting trace data, click the **Start With** button.

For more information about the Microsoft Message Analyzer **Web Proxy** trace, see [Microsoft-Pef-WebProxy Provider](#).

The built-in **Web Proxy** trace in Microsoft Message Analyzer is based on Fiddler; it can capture client-side HTTPS traffic and display unencrypted HTTPS messages. The **Web Proxy** trace works by configuring a local proxy for all HTTP and HTTPS traffic that gives it access to unencrypted messages.

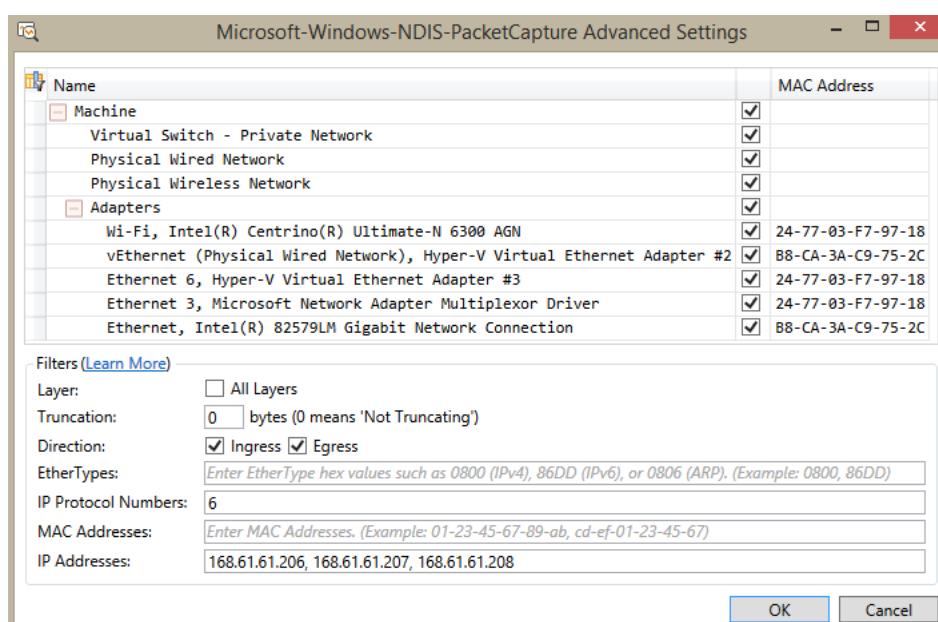
#### Diagnosing network issues using Microsoft Message Analyzer

In addition to using the Microsoft Message Analyzer **Web Proxy** trace to capture details of the HTTP/HTTPs traffic between the client application and the storage service, you can also use the built-in **Local Link Layer** trace to capture network packet information. This enables you to capture data similar to that which you can capture with Wireshark, and diagnose network issues such as dropped packets.

The following screenshot shows an example **Local Link Layer** trace with some **informational** messages in the **DiagnosisTypes** column. Clicking on an icon in the **DiagnosisTypes** column shows the details of the message. In this example, the server retransmitted message #305 because it did not receive an acknowledgement from the client:

MessageNumber	Type	Timestamp	TimeElapsed	Source
302	i	2014-06-27T11:09:05.5912057	0	192.168.0.16
Type Level Message				
303	i	2014-06-27T11:09:05.6019881	0	192.168.0.16
304	i	2014-06-27T11:09:05.6020185	0	192.168.0.16
305	i	2014-06-27T11:09:05.6270940	0	168.61.61.207
306	i	2014-06-27T11:09:05.6270947	0.0001652	168.61.61.207
307	i	2014-06-27T11:09:05.6271621	0	168.61.61.207
Fields Stack Diagnosis				
308	i	2014-06-27T11:09:05.6271629	0	168.61.61.207
310	i	2014-06-27T11:09:05.6272789	0	192.168.0.16

When you create the trace session in Microsoft Message Analyzer, you can specify filters to reduce the amount of noise in the trace. On the **Capture / Trace** page where you define the trace, click on the **Configure** link next to **Microsoft-Windows-NDIS-PacketCapture**. The following screenshot shows a configuration that filters TCP traffic for the IP addresses of three storage services:



For more information about the Microsoft Message Analyzer Local Link Layer trace, see [Microsoft-Pef-NDIS-PacketCapture Provider](#).

#### **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](#)

# Azure Storage metrics in Azure Monitor

3/25/2020 • 13 minutes to read • [Edit Online](#)

With metrics on Azure Storage, you can analyze usage trends, trace requests, and diagnose issues with your storage account.

Azure Monitor provides unified user interfaces for monitoring across different Azure services. For more information, see [Azure Monitor](#). Azure Storage integrates Azure Monitor by sending metric data to the Azure Monitor platform.

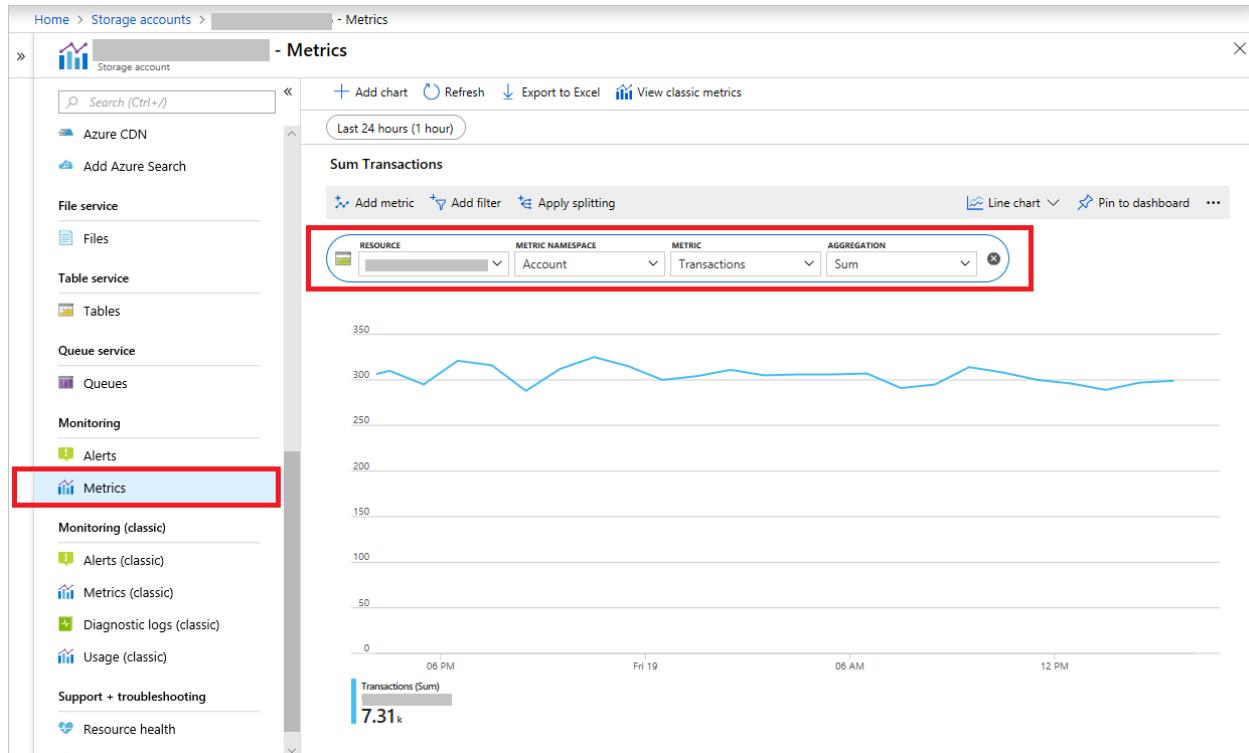
## Access metrics

Azure Monitor provides multiple ways to access metrics. You can access them from the [Azure portal](#), the Azure Monitor APIs (REST, and .NET) and analysis solutions such as Event Hubs. For more information, see [Azure Monitor Metrics](#).

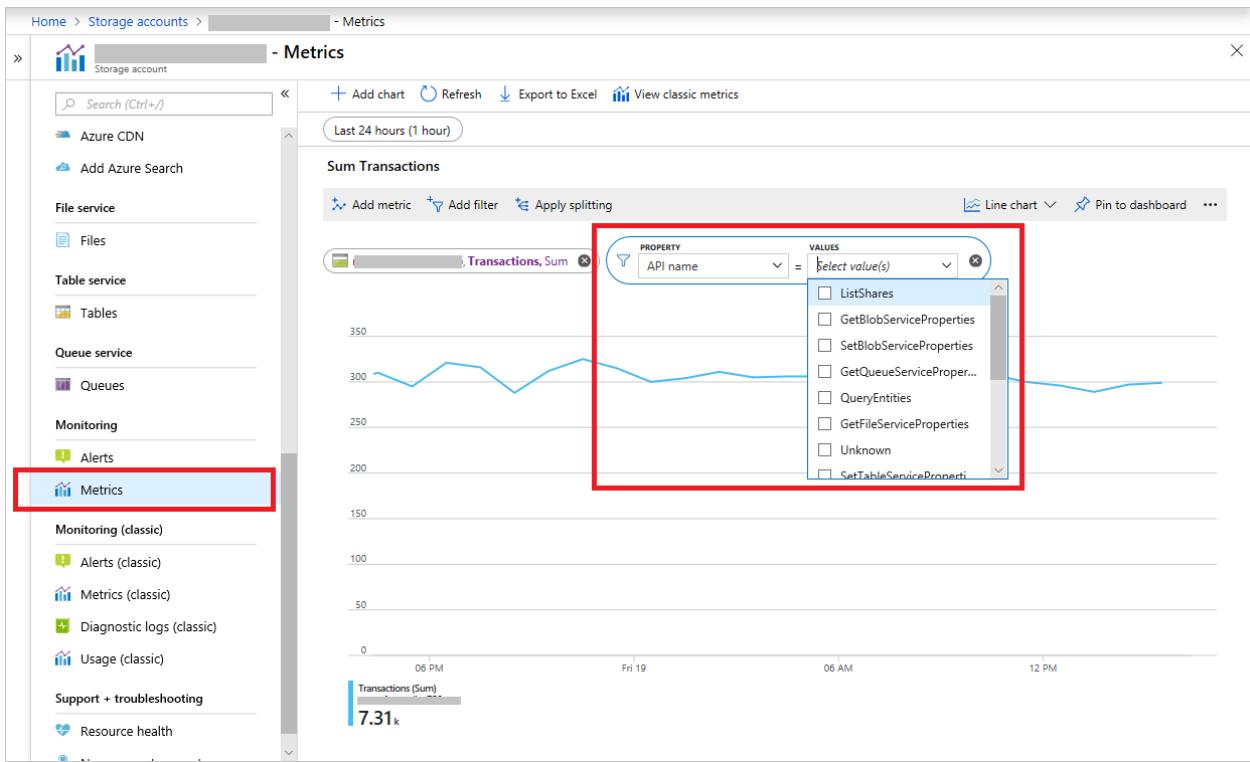
Metrics are enabled by default, and you can access the past 93 days of data. If you need to retain data for a longer period of time, you can archive metrics data to an Azure Storage account. This is configured in [diagnostic settings](#) in Azure Monitor.

### Access metrics in the Azure portal

You can monitor metrics over time in the Azure portal. The following example shows how to view **Transactions** at account level.



For metrics supporting dimensions, you can filter metric with the desired dimension value. The following example shows how to view **Transactions** at account level on a specific operation by selecting values for **API Name** dimension.



## Access metrics with the REST API

Azure Monitor provides [REST APIs](#) to read metric definition and values. This section shows you how to read the storage metrics. Resource ID is used in all REST APIs. For more information, please read [Understanding resource ID for services in Storage](#).

The following example shows how to use [ArmClient](#) at the command line to simplify testing with the REST API.

### List account level metric definition with the REST API

The following example shows how to list metric definition at account level:

```
Login to Azure and enter your credentials when prompted.
> armclient login

> armclient GET
/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccoun
ts/{storageAccountName}/providers/microsoft.insights/metricdefinitions?api-version=2018-01-01
```

If you want to list the metric definitions for blob, table, file, or queue, you must specify different resource IDs for each service with the API.

The response contains the metric definition in JSON format:

```
{
 "value": [
 {
 "id": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/{storageAccountName}/providers/microsoft.insights/metricdefinitions/UsedCapacity",
 "resourceId": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/{storageAccountName}",
 "category": "Capacity",
 "name": {
 "value": "UsedCapacity",
 "localizedValue": "Used capacity"
 },
 "isDimensionRequired": false,
 "unit": "Bytes",
 "primaryAggregationType": "Average",
 "metricAvailabilities": [
 {
 "timeGrain": "PT1M",
 "retention": "P30D"
 },
 {
 "timeGrain": "PT1H",
 "retention": "P30D"
 }
],
 ...
 next metric definition
]
 }
}
```

#### **Read account-level metric values with the REST API**

The following example shows how to read metric data at account level:

```
> armclient GET
"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/{storageAccountName}/providers/microsoft.insights/metrics?metricnames=Availability&api-version=2018-01-01&aggregation=Average&interval=PT1H"
```

In above example, if you want to read metric values for blob, table, file, or queue, you must specify different resource IDs for each service with the API.

The following response contains metric values in JSON format:

```
{
 "cost": 0,
 "timespan": "2017-09-07T17:27:41Z/2017-09-07T18:27:41Z",
 "interval": "PT1H",
 "value": [
 {
 "id":
 "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/{storageAccountName}/providers/Microsoft.Insights/metrics/Availability",
 "type": "Microsoft.Insights/metrics",
 "name": {
 "value": "Availability",
 "localizedValue": "Availability"
 },
 "unit": "Percent",
 "timeseries": [
 {
 "metadatavalues": [],
 "data": [
 {
 "timeStamp": "2017-09-07T17:27:00Z",
 "average": 100.0
 }
]
 }
]
 }
]
}
```

## Access metrics with the .NET SDK

Azure Monitor provides [.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 later version for storage metrics. Resource ID is used in .NET SDK. For more information, please read [Understanding resource ID for services in Storage](#).

The following example shows how to use Azure Monitor .NET SDK to read storage metrics.

### List account level metric definition with the .NET SDK

The following example shows how to list metric definition at account level:

```

public static async Task ListStorageMetricDefinition()
{
 // Resource ID for storage account
 var resourceId =
 "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/{storageAccountName}";
 var subscriptionId = "{SubscriptionID}";
 // How to identify Tenant ID, Application ID and Access Key:
 https://azure.microsoft.com/documentation/articles/resource-group-create-service-principal-portal/
 var tenantId = "{TenantID}";
 var applicationId = "{ApplicationID}";
 var accessKey = "{AccessKey}";

 // Using metrics in Azure Monitor is currently free. However, if you use additional solutions
 // ingesting metrics data, you may be billed by these solutions. For example, you are billed by Azure Storage
 // if you archive metrics data to an Azure Storage account. Or you are billed by Operation Management Suite
 // (OMS) if you stream metrics data to OMS for advanced analysis.

 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)
 {
 //Enumrate metric definition:
 // Id
 // ResourceId
 // Name
 // Unit
 // MetricAvailabilities
 // PrimaryAggregationType
 // Dimensions
 // IsDimensionRequired
 }
}

```

If you want to list the metric definitions for blob, table, file, or queue, you must specify different resource IDs for each service with the API.

#### **Read metric values with the .NET SDK**

The following example shows how to read `UsedCapacity` data at account level:

```

public static async Task ReadStorageMetricValue()
{
 // Resource ID for storage account
 var resourceId =
 "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/{storageAccountName}";
 var subscriptionId = "{SubscriptionID}";
 // How to identify Tenant ID, Application ID and Access Key:
 https://azure.microsoft.com/documentation/articles/resource-group-create-service-principal-portal/
 var tenantId = "{TenantID}";
 var applicationId = "{ApplicationID}";
 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)
 {
 //Enumrate metric value
 // Id
 // Name
 // Type
 // Unit
 // Timeseries
 // - Data
 // - Metadatavalues
 }
}

```

In above example, if you want to read metric values for blob, table, file, or queue, you must specify different resource IDs for each service with the API.

#### **Read multi-dimensional metric values with the .NET SDK**

For multi-dimensional metrics, you need to define meta data filter if you want to read metric data on specific dimension value.

The following example shows how to read metric data on the metric supporting multi-dimension:

```

public static async Task ReadStorageMetricValueTest()
{
 // Resource ID for blob storage
 var resourceId =
 "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/{storageAccountName}/blobServices/default";
 var subscriptionId = "{SubscriptionID}";
 // How to identify Tenant ID, Application ID and Access Key:
 https://azure.microsoft.com/documentation/articles/resource-group-create-service-principal-portal/
 var tenantId = "{TenantID}";
 var applicationId = "{ApplicationID}";
 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)
 {
 //Enumrate metric value
 // Id
 // Name
 // Type
 // Unit
 // Timeseries
 // - Data
 // - Metadatavalues
 }
}

```

## Understanding resource ID for services in Azure Storage

Resource ID is a unique identifier of a resource in Azure. When you use the Azure Monitor REST API to read metrics definitions or values, you must use resource ID for the resource on which you intend to operate. The resource ID template follows this format:

```
/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/{resourceProviderNamespace}/{re
sourceType}/{resourceName}
```

Storage provides metrics at both the storage account level and the service level with Azure Monitor. For example, you can retrieve metrics for just Blob storage. Each level has its own resource ID, which is used to retrieve the metrics for just that level.

## Resource ID for a storage account

The following shows the format for specifying the Resource ID for a storage account.

```
/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/{storageAccountName}
```

## Resource ID for the storage services

The following shows the format for specifying the Resource ID for each of the storage services.

- Blob service resource ID

```
/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/{storageAccountName}/blobServices/default
```

- Table service resource ID

```
/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/{storageAccountName}/tableServices/default
```

- Queue service resource ID

```
/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/{storageAccountName}/queueServices/default
```

- File service resource ID

```
/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/{storageAccountName}/fileServices/default
```

## Resource ID in Azure Monitor REST API

The following shows the pattern used when calling the Azure Monitor REST API.

```
GET {resourceId}/providers/microsoft.insights/metrics?{parameters}
```

## Capacity metrics

Capacity metrics values are sent to Azure Monitor every hour. The values are refreshed daily. 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

METRIC NAME	DESCRIPTION
-------------	-------------

METRIC NAME	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

METRIC NAME	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="#">BlobTier</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="#">BlobTier</a> (<a href="#">Definition</a>)</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>

## Table storage

METRIC NAME	DESCRIPTION
TableCapacity	<p>The amount of Table storage used by the storage account.</p> <p>Unit: Bytes Aggregation Type: Average Value example: 1024</p>
TableCount	<p>The number of tables in the storage account.</p> <p>Unit: Count Aggregation Type: Average Value example: 1024</p>

METRIC NAME	DESCRIPTION
TableEntityCount	The number of table entities in the storage account.  Unit: Count Aggregation Type: Average Value example: 1024

## Queue storage

METRIC NAME	DESCRIPTION
QueueCapacity	The amount of Queue storage used by the storage account.  Unit: Bytes Aggregation Type: Average Value example: 1024
QueueCount	The number of queues in the storage account.  Unit: Count Aggregation Type: Average Value example: 1024
QueueMessageCount	The number of unexpired queue messages in the storage account.  Unit: Count Aggregation Type: Average Value example: 1024

## File storage

METRIC NAME	DESCRIPTION
FileCapacity	The amount of File storage used by the storage account.  Unit: Bytes Aggregation Type: Average Value example: 1024
FileCount	The number of files in the storage account.  Unit: Count Aggregation Type: Average Value example: 1024
FileShareCount	The number of file shares in the storage account.  Unit: Count Aggregation Type: Average Value example: 1024

## 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 service level (Blob storage, Table storage, Azure Files, and

Queue storage). 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 NAME	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: ResponseType, GeoType, ApiName, and Authentication (<a href="#">Definition</a>)            Value example: 1024</p>
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 to an external client from 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.</p> <p>Unit: Milliseconds            Aggregation Type: Average            Applicable dimensions: GeoType, ApiName, and Authentication (<a href="#">Definition</a>)            Value example: 1024</p>

METRIC NAME	DESCRIPTION
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.

DIMENSION NAME	DESCRIPTION
<b>BlobType</b>	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 Blob is included in BlockBlob.
<b>BlobTier</b>	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>• <b>Hot</b>: Hot tier</li> <li>• <b>Cool</b>: Cool tier</li> <li>• <b>Archive</b>: Archive tier</li> <li>• <b>Premium</b>: Premium tier for block blob</li> <li>• <b>P4/P6/P10/P15/P20/P30/P40/P50/P60</b>: Tier types for premium page blob</li> <li>• <b>Standard</b>: Tier type for standard page Blob</li> <li>• <b>Untiered</b>: Tier type for general purpose v1 storage account</li> </ul>
<b>GeoType</b>	Transaction from Primary or Secondary cluster. The available values include <b>Primary</b> and <b>Secondary</b> . 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>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>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> </ul>
ApiName	<p>The name of operation. For example:</p> <ul style="list-style-type: none"> <li>• <b>CreateContainer</b></li> <li>• <b>DeleteBlob</b></li> <li>• <b>GetBlob</b></li> </ul> <p>For all operation names, see <a href="#">document</a>.</p>
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>

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**. Or if you look at **BlobCount** value for Block Blob, you need to filter the **BlobType** dimension with **BlockBlob**.

# Service continuity of legacy metrics

Legacy metrics are available in parallel with Azure Monitor managed metrics. The support keeps the same until Azure Storage ends the service on legacy metrics.

## FAQ

### **Does new metrics support Classic Storage account?**

No, new metrics in Azure Monitor only support Azure Resource Manager storage accounts. If you want to use metrics on Storage accounts, you need to migrate to Azure Resource Manager Storage account. See [Migrate to Azure Resource Manager](#).

### **Does Azure Storage support metrics for Managed Disks or Unmanaged Disks?**

No, Azure Compute supports the metrics on disks. See [article](#) for more details.

### **How to map and migrate classic metrics with new metrics?**

You can find detailed mapping between classic metrics and new metrics in [Azure Storage metrics migration](#).

## Next steps

- [Azure Monitor](#)

# Azure Storage metrics migration

8/7/2019 • 4 minutes to read • [Edit Online](#)

Aligned with the strategy of unifying the monitor experience in Azure, Azure Storage integrates metrics to the Azure Monitor platform. In the future, the service of the old metrics will end with an early notification based on Azure policy. If you rely on old storage metrics, you need to migrate prior to the service end date in order to maintain your metric information.

This article shows you how to migrate from the old metrics to the new metrics.

## Understand old metrics that are managed by Azure Storage

Azure Storage collects old metric values, and aggregates and stores them in \$Metric tables within the same storage account. You can use the Azure portal to set up a monitoring chart. You can also use the Azure Storage SDKs to read the data from \$Metric tables that are based on the schema. For more information, see [Storage Analytics](#).

Old metrics provide capacity metrics only on Azure Blob storage. Old metrics provide transaction metrics on Blob storage, Table storage, Azure Files, and Queue storage.

Old metrics are designed in a flat schema. The design results in zero metric value when you don't have the traffic patterns triggering the metric. For example, the **ServerTimeoutError** value is set to 0 in \$Metric tables even when you don't receive any server timeout errors from the live traffic to a storage account.

## Understand new metrics managed by Azure Monitor

For new storage metrics, Azure Storage emits the metric data to the Azure Monitor back end. Azure Monitor provides a unified monitoring experience, including data from the portal as well as data ingestion. For more details, you can refer to this [article](#).

New metrics provide capacity metrics and transaction metrics on Blob, Table, File, Queue, and premium storage.

Multi-dimension is one of the features that Azure Monitor provides. Azure Storage adopts the design in defining new metric schema. For supported dimensions on metrics, you can find details in [Azure Storage metrics in Azure Monitor](#). Multi-dimension design provides cost efficiency on both bandwidth from ingestion and capacity from storing metrics. Consequently, if your traffic has not triggered related metrics, the related metric data will not be generated. For example, if your traffic has not triggered any server timeout errors, Azure Monitor doesn't return any data when you query the value of metric **Transactions** with dimension **ResponseType** equal to **ServerTimeoutError**.

## Metrics mapping between old metrics and new metrics

If you read metric data programmatically, you need to adopt the new metric schema in your programs. To better understand the changes, you can refer to the mapping listed in the following table:

### Capacity metrics

OLD METRIC	NEW METRIC
Capacity	BlobCapacity with the dimension <b>BlobType</b> equal to <b>BlockBlob</b> or <b>PageBlob</b>

OLD METRIC	NEW METRIC
ObjectCount	BlobCount with the dimension <b>BlobType</b> equal to <b>BlockBlob</b> or <b>PageBlob</b>
ContainerCount	ContainerCount

The following metrics are new offerings that the old metrics don't support:

- **TableCapacity**
- **TableCount**
- **TableEntityCount**
- **QueueCapacity**
- **QueueCount**
- **QueueMessageCount**
- **FileCapacity**
- **FileCount**
- **FileShareCount**
- **UsedCapacity**

#### Transaction metrics

OLD METRIC	NEW METRIC
<b>AnonymousAuthorizationError</b>	Transactions with the dimension <b>ResponseType</b> equal to <b>AuthorizationError</b> and dimension <b>Authentication</b> equal to <b>Anonymous</b>
<b>AnonymousClientOtherError</b>	Transactions with the dimension <b>ResponseType</b> equal to <b>ClientOtherError</b> and dimension <b>Authentication</b> equal to <b>Anonymous</b>
<b>AnonymousClientTimeoutError</b>	Transactions with the dimension <b>ResponseType</b> equal to <b>ClientTimeoutError</b> and dimension <b>Authentication</b> equal to <b>Anonymous</b>
<b>AnonymousNetworkError</b>	Transactions with the dimension <b>ResponseType</b> equal to <b>NetworkError</b> and dimension <b>Authentication</b> equal to <b>Anonymous</b>
<b>AnonymousServerOtherError</b>	Transactions with the dimension <b>ResponseType</b> equal to <b>ServerOtherError</b> and dimension <b>Authentication</b> equal to <b>Anonymous</b>
<b>AnonymousServerTimeoutError</b>	Transactions with the dimension <b>ResponseType</b> equal to <b>ServerTimeoutError</b> and dimension <b>Authentication</b> equal to <b>Anonymous</b>
<b>AnonymousSuccess</b>	Transactions with the dimension <b>ResponseType</b> equal to <b>Success</b> and dimension <b>Authentication</b> equal to <b>Anonymous</b>
<b>AnonymousThrottlingError</b>	Transactions with the dimension <b>ResponseType</b> equal to <b>ClientThrottlingError</b> or <b>ServerBusyError</b> and dimension <b>Authentication</b> equal to <b>Anonymous</b>

OLD METRIC	NEW METRIC
AuthorizationError	Transactions with the dimension <b>ResponseType</b> equal to <b>AuthorizationError</b>
Availability	Availability
AverageE2ELatency	SuccessE2ELatency
AverageServerLatency	SuccessServerLatency
ClientOtherError	Transactions with the dimension <b>ResponseType</b> equal to <b>ClientOtherError</b>
ClientTimeoutError	Transactions with the dimension <b>ResponseType</b> equal to <b>ClientTimeoutError</b>
NetworkError	Transactions with the dimension <b>ResponseType</b> equal to <b>NetworkError</b>
PercentAuthorizationError	Transactions with the dimension <b>ResponseType</b> equal to <b>AuthorizationError</b>
PercentClientOtherError	Transactions with the dimension <b>ResponseType</b> equal to <b>ClientOtherError</b>
PercentNetworkError	Transactions with the dimension <b>ResponseType</b> equal to <b>NetworkError</b>
PercentServerOtherError	Transactions with the dimension <b>ResponseType</b> equal to <b>ServerOtherError</b>
PercentSuccess	Transactions with the dimension <b>ResponseType</b> equal to <b>Success</b>
PercentThrottlingError	Transactions with the dimension <b>ResponseType</b> equal to <b>ClientThrottlingError</b> or <b>ServerBusyError</b>
PercentTimeoutError	Transactions with the dimension <b>ResponseType</b> equal to <b>ServerTimeoutError</b> or <b>ResponseType</b> equal to <b>ClientTimeoutError</b>
SASAuthorizationError	Transactions with the dimension <b>ResponseType</b> equal to <b>AuthorizationError</b> and dimension <b>Authentication</b> equal to <b>SAS</b>
SASClientOtherError	Transactions with the dimension <b>ResponseType</b> equal to <b>ClientOtherError</b> and dimension <b>Authentication</b> equal to <b>SAS</b>
SASClientTimeoutError	Transactions with the dimension <b>ResponseType</b> equal to <b>ClientTimeoutError</b> and dimension <b>Authentication</b> equal to <b>SAS</b>
SASNetworkError	Transactions with the dimension <b>ResponseType</b> equal to <b>NetworkError</b> and dimension <b>Authentication</b> equal to <b>SAS</b>

OLD METRIC	NEW METRIC
SASServerOtherError	Transactions with the dimension <b>ResponseType</b> equal to <b>ServerOtherError</b> and dimension <b>Authentication</b> equal to <b>SAS</b>
SASServerTimeoutError	Transactions with the dimension <b>ResponseType</b> equal to <b>ServerTimeoutError</b> and dimension <b>Authentication</b> equal to <b>SAS</b>
SASSuccess	Transactions with the dimension <b>ResponseType</b> equal to <b>Success</b> and dimension <b>Authentication</b> equal to <b>SAS</b>
SASThrottlingError	Transactions with the dimension <b>ResponseType</b> equal to <b>ClientThrottlingError</b> or <b>ServerBusyError</b> and dimension <b>Authentication</b> equal to <b>SAS</b>
ServerOtherError	Transactions with the dimension <b>ResponseType</b> equal to <b>ServerOtherError</b>
ServerTimeoutError	Transactions with the dimension <b>ResponseType</b> equal to <b>ServerTimeoutError</b>
Success	Transactions with the dimension <b>ResponseType</b> equal to <b>Success</b>
ThrottlingError	Transactions with the dimension <b>ResponseType</b> equal to <b>ClientThrottlingError</b> or <b>ServerBusyError</b>
TotalBillableRequests	Transactions
TotalEgress	Egress
TotalIngress	Ingress
TotalRequests	Transactions

## FAQ

### How should I migrate existing alert rules?

If you have created classic alert rules based on old storage metrics, you need to create new alert rules based on the new metric schema.

### Is new metric data stored in the same storage account by default?

No. To archive the metric data to a storage account, use the [Azure Monitor Diagnostic Setting API](#).

## Next steps

- [Azure Monitor](#)
- [Storage metrics in Azure Monitor](#)

# Azure Storage analytics logging

4/3/2020 • 8 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.

Storage Analytics logging is not enabled by default for your storage account. You can enable it in 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 [Get Blob Service Properties](#), [Get Queue Service Properties](#), and [Get Table Service Properties](#) operations to enable Storage Analytics for each service.

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. However, premium storage account is not supported.

## 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.

## 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-AzureStorageBlob -Container '$logs' |
Where-Object {
 $_.Name -match 'table/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>

ATTRIBUTE	DESCRIPTION
MM	The two digit month for the log. For example: 07
DD	The two digit day for the log. For example: 31
hh	The two digit hour that indicates the starting hour for the logs, in 24 hour UTC format. For example: 18
mm	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 00.
<counter>	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 000000. For example: 000001

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
LogType	<p>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.</p> <p>Example 1: write</p> <p>Example 2: read,write</p> <p>Example 3: read,write,delete</p>
StartTime	<p>The earliest time of an entry in the log, in the form of YYYY-MM-DDThh:mm:ssZ. For example: 2011-07-31T18:21:46Z</p>
EndTime	<p>The latest time of an entry in the log, in the form of YYYY-MM-DDThh:mm:ssZ. For example: 2011-07-31T18:22:09Z</p>

ATTRIBUTE	DESCRIPTION
LogVersion	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

## Enable Storage logging

You can enable Storage logging with Azure portal, PowerShell, and Storage SDKs.

### Enable Storage logging using the Azure portal

In the Azure portal, use the **Diagnostics settings (classic)** blade to control Storage Logging, accessible from the **Monitoring (classic)** section of a storage account's **Menu blade**.

You can specify the storage services that you want to log, and the retention period (in days) for the logged data.

### Enable Storage logging using PowerShell

You can use PowerShell on your local machine to configure Storage Logging in your storage account by using the Azure PowerShell cmdlet **Get-AzureStorageServiceLoggingProperty** to retrieve the current settings, and the cmdlet **Set-AzureStorageServiceLoggingProperty** to change the current settings.

The cmdlets that control Storage Logging use a **LoggingOperations** parameter that is a string containing a comma-separated list of request types to log. The three possible request types are **read**, **write**, and **delete**. To switch off logging, use the value **none** for the **LoggingOperations** parameter.

The following command switches on logging for read, write, and delete requests in the Queue service in your default storage account with retention set to five days:

```
Set-AzureStorageServiceLoggingProperty -ServiceType Queue -LoggingOperations read,write,delete -RetentionDays 5
```

The following command switches off logging for the table service in your default storage account:

```
Set-AzureStorageServiceLoggingProperty -ServiceType Table -LoggingOperations none
```

For information about how to configure the Azure PowerShell cmdlets to work with your Azure subscription and how to select the default storage account to use, see: [How to install and configure Azure PowerShell](#).

### Enable Storage logging programmatically

In addition to using the Azure portal or the Azure PowerShell cmdlets to control Storage Logging, you can also use one of the Azure Storage APIs. For example, if you are using a .NET language you can use the Storage Client Library.

The classes **CloudBlobClient**, **CloudQueueClient**, and **CloudTableClient** all have methods such as **SetServiceProperties** and **SetServicePropertiesAsync** that take a **ServiceProperties** object as a parameter. You can use the **ServiceProperties** object to configure Storage Logging. For example, the following C# snippet shows how to change what is logged and the retention period for queue logging:

```
var storageAccount = CloudStorageAccount.Parse(connStr);
var queueClient = storageAccount.CreateCloudQueueClient();
var serviceProperties = queueClient.GetServiceProperties();

serviceProperties.Logging.LoggingOperations = LoggingOperations.All;
serviceProperties.Logging.RetentionDays = 2;

queueClient.SetServiceProperties(serviceProperties);
```

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](#).

## Download Storage logging 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 specific files](#).

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, including Microsoft Message Analyzer (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

- [Storage Analytics Log Format](#)

- Storage Analytics Logged Operations and Status Messages
- Storage Analytics Metrics (classic)

# Create an Azure Storage account

2/28/2020 • 8 minutes to read • [Edit Online](#)

An Azure storage account contains all of your Azure Storage data objects: blobs, files, queues, tables, and disks. The storage account provides a unique namespace for your Azure Storage data that is accessible from anywhere in the world over HTTP or HTTPS. Data in your Azure storage account is durable and highly available, secure, and massively scalable.

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](#).

## NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

## Prerequisites

If you don't have an Azure subscription, create a [free account](#) before you begin.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [Template](#)

None.

## Sign in to Azure

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [Template](#)

Sign in to the [Azure portal](#).

## Create a storage account

Now you are ready to create a storage account.

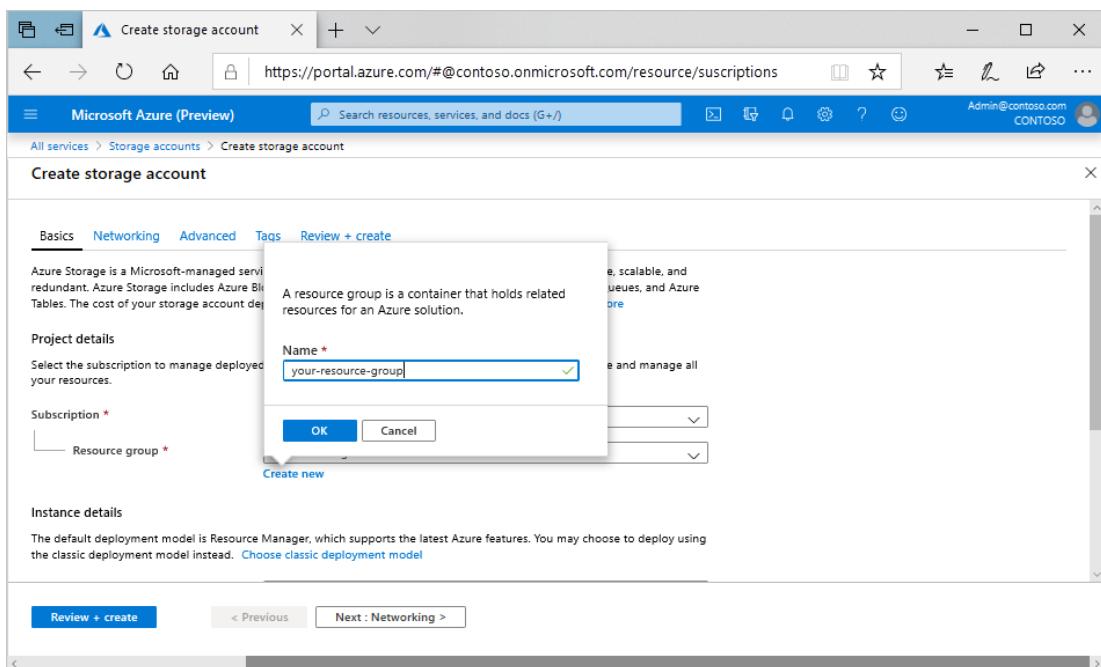
Every storage account must belong to an Azure resource group. A resource group is a logical container for grouping your Azure services. When you create a storage account, you have the option to either create a new resource group, or use an existing resource group. This article shows how to create a new resource group.

A **general-purpose v2** storage account provides access to all of the Azure Storage services: blobs, files, queues, tables, and disks. The steps outlined here create a general-purpose v2 storage account, but the steps to create any type of storage account are similar.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [Template](#)

To create a general-purpose v2 storage account in the Azure portal, follow these steps:

1. On the Azure portal menu, select **All services**. In the list of resources, type **Storage Accounts**. As you begin typing, the list filters based on your input. Select **Storage Accounts**.
2. On the **Storage Accounts** window that appears, choose **Add**.
3. Select the subscription in which to create the storage account.
4. Under the **Resource group** field, select **Create new**. Enter a name for your new resource group, as shown in the following image.



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 can include numbers and lowercase letters only.
6. Select a location for your storage account, or use the default location.
7. Leave these fields set to their default values:

FIELD	VALUE
Deployment model	Resource Manager
Performance	Standard
Account kind	StorageV2 (general-purpose v2)
Replication	Read-access geo-redundant storage (RA-GRS)
Access tier	Hot

8. If you plan to use [Azure Data Lake Storage](#), choose the **Advanced** tab, and then set **Hierarchical**

namespace to Enabled.

9. Select **Review + Create** to review your storage account settings and create the account.

10. Select **Create**.

For more information about types of storage accounts and other storage account settings, see [Azure storage account overview](#). For more information on resource groups, see [Azure Resource Manager overview](#).

For more information about available replication options, see [Storage replication options](#).

## Delete a storage account

Deleting a storage account deletes the entire account, including all data in the account, and cannot be undone.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [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](#).

### WARNING

It's not possible to restore a deleted storage account or retrieve any of the content that it contained before deletion. Be sure to back up anything you want to save before you delete the account. This also holds true for any resources in the account—once you delete a blob, table, queue, or file, it is permanently deleted.

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](#).

## Next steps

In this how-to article, you've created a general-purpose v2 standard storage account. To learn how to upload and download blobs to and from your storage account, continue to one of the Blob storage quickstarts.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)
- [Template](#)

[Work with blobs using the Azure portal](#)

# Migrate Azure Data Lake Storage from Gen1 to Gen2

4/20/2020 • 6 minutes to read • [Edit Online](#)

You can migrate your data, workloads, and applications from Data Lake Storage Gen1 to Data Lake Storage Gen2.

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

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.

## Recommended approach

To migrate 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

## NOTE

Gen1 and Gen2 are different services, there is no in-place upgrade experience, intentional migration effort required.

### 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.

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.
4. Update applications to use Gen2 APIs. See guides for [.NET](#), [Java](#), [Python](#), [JavaScript](#) and [REST](#).
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 [Role-based access control \(RBAC\) 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	AAD managed identity Service principals	AAD managed identity Service principals Shared Access Key
Authorization	Management - RBAC Data – ACLs	Management – RBAC Data - ACLs, RBAC
Encryption – Data at rest	Server side – with Microsoft-managed or customer-managed keys	Server side – with Microsoft-managed or customer-managed keys

Area	Gen1	Gen2
VNET Support	VNET Integration	Service Endpoints, Private Endpoints
Developer experience	REST, .NET, Java, Python, PowerShell, Azure CLI	Generally available - REST, .NET, Java, Python Public preview - JavaScript, PowerShell, Azure CLI
Diagnostic logs	Classic logs Azure Monitor integrated	Classic logs - Generally available Azure monitor integration – timeline TBD
Ecosystem	HDInsight (3.6), Azure Databricks (3.1 and above), SQL DW, ADF	HDInsight (3.6, 4.0), Azure Databricks (5.1 and above), SQL DW, ADF

## Gen1 to Gen2 patterns

Choose a migration pattern, and then modify that pattern as needed.

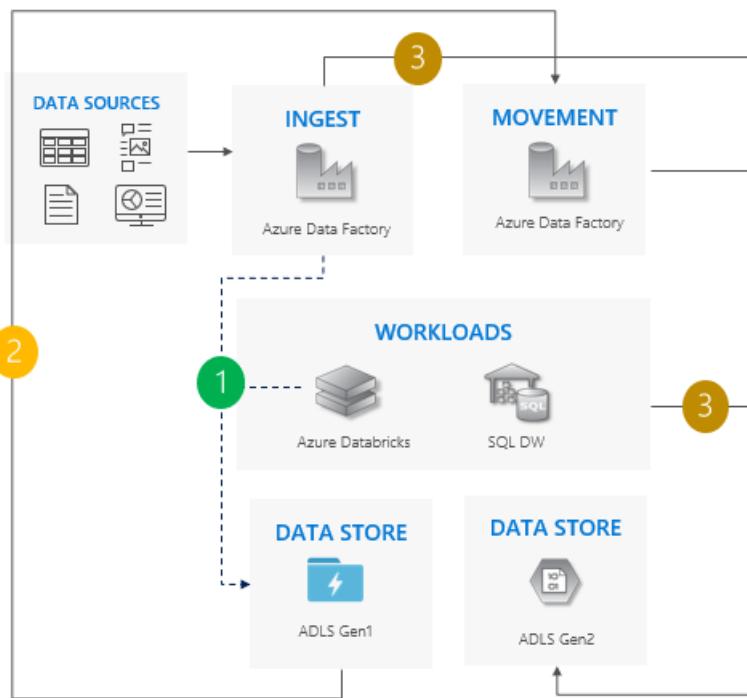
Lift and Shift	The simplest pattern. Ideal if your data pipelines can afford downtime.
Incremental copy	Similar to <i>lift and shift</i> , but with less downtime. Ideal for large amounts of data that take longer to copy.
Dual pipeline	Ideal for pipelines that can't afford any downtime.
Bidirectional 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](#). ACLs copy with the data.
3. Point ingest operations and workloads to Gen2.
4. Decommission Gen1.

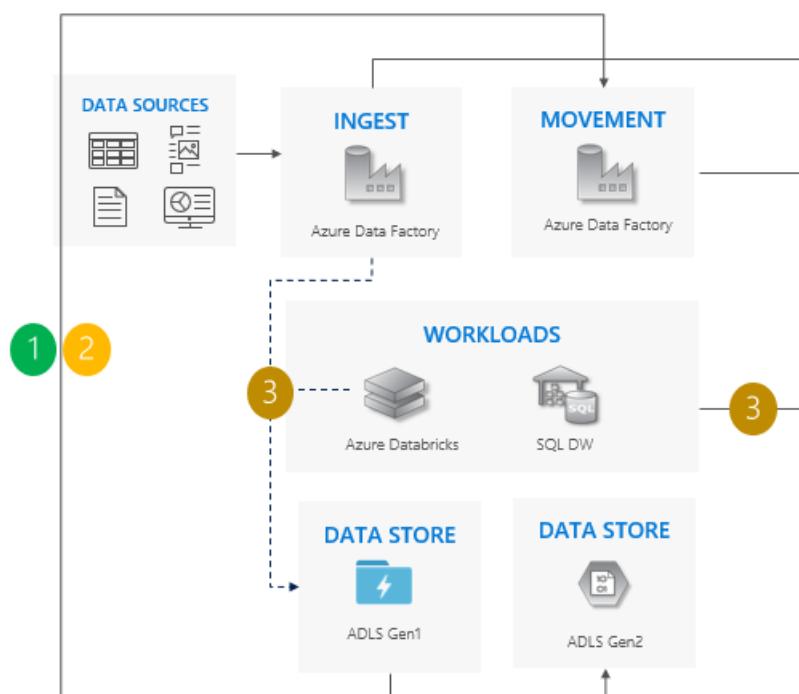


#### Considerations for using the lift and shift pattern

- ✓ 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.

#### 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.

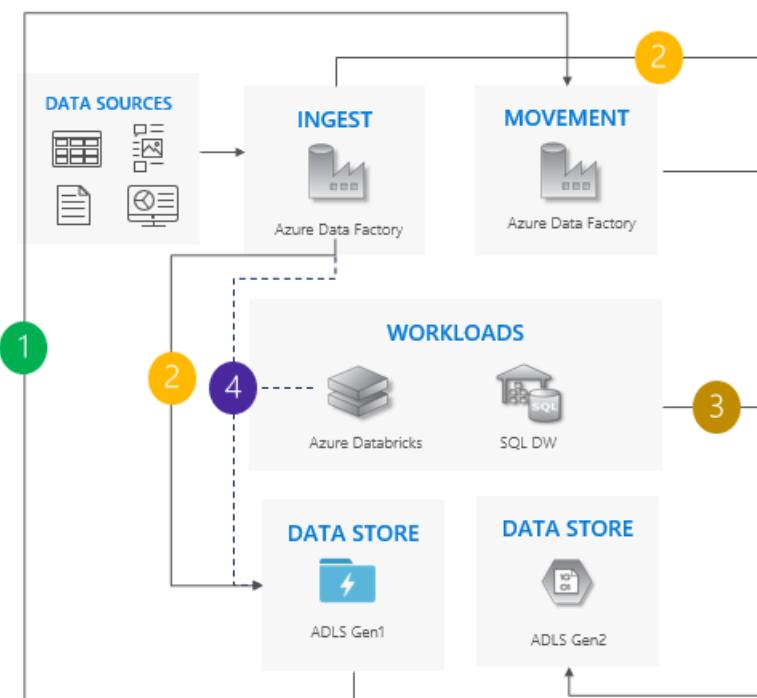


#### Considerations for using the incremental copy pattern:

- ✓ 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.

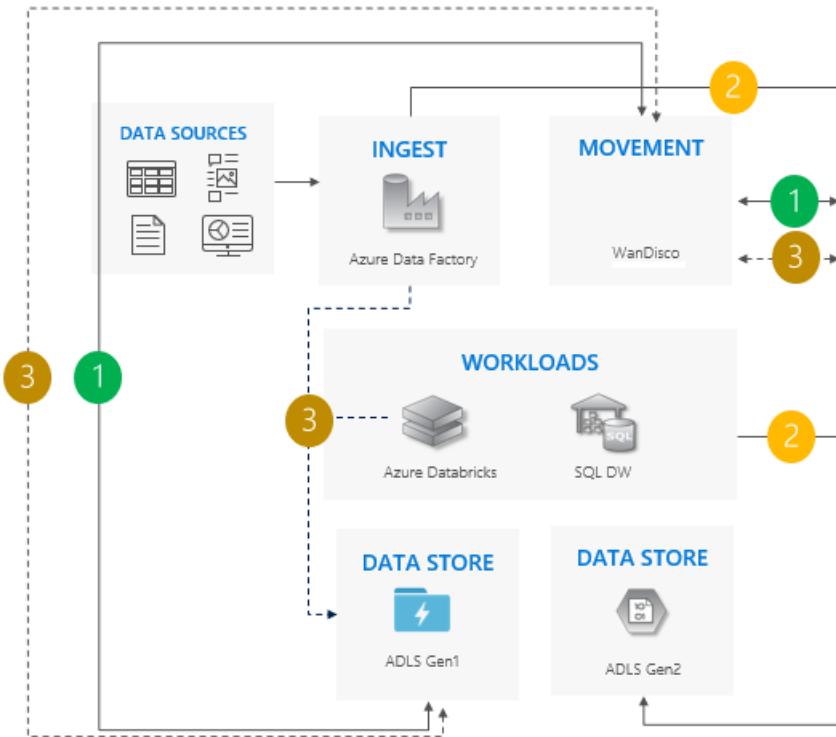


#### Considerations for using the dual pipeline pattern:

- ✓ 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.



#### 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. 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](#)

# Migrate from on-prem HDFS store to Azure Storage with Azure Data Box

3/4/2020 • 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 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 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.

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.mydatabboxno.microsoftdatabox.com=myaccountkey \
-filter ./exclusions.lst -f /tmp/copylist1 -m 4 \
/data/testfiles \
wasb://hdfscontainer@mystorageaccount.blob.mydatabboxno.microsoftdatabox.com/data
```

To improve the copy speed:

- Try changing the number of mappers. (The above example uses `m` = 4 mappers.)
- Try running multiple `distcp` in parallel.
- Remember that large files perform better than small files.

## 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. See [Use Azure Data Lake Storage Gen2 with Azure HDInsight clusters](#).

# Use the Azure portal to access blob or queue data

4/14/2020 • 6 minutes to read • [Edit Online](#)

When you access blob or queue 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 or queue data

Depending on how you want to authorize access to blob or queue data in the Azure portal, you'll need specific permissions. In most cases, these permissions are provided via role-based access control (RBAC). For more information about RBAC, see [What is role-based access control \(RBAC\)?](#).

### Use the account access key

To access blob and queue data with the account access key, you must have an RBAC role assigned to you that includes the RBAC action **Microsoft.Storage/storageAccounts/listkeys/action**. This RBAC role may be a built-in or a custom role. Built-in roles that support **Microsoft.Storage/storageAccounts/listkeys/action** include:

- The Azure Resource Manager [Owner](#) role
- The Azure Resource Manager [Contributor](#) role
- The [Storage Account Contributor](#) role

When you attempt to access blob or queue 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 and queue data. If you have not been assigned a role with this action, then the portal attempts to access data using 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 and queue data with the account key. For more information, see [Classic subscription administrator roles, Azure RBAC roles, and Azure AD administrator roles](#).

### Use your Azure AD account

To access blob or queue data from the Azure portal using your Azure AD account, both of the following statements must be true for you:

- 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.
- You have been assigned either a built-in or custom role that provides access to blob or queue data.

The [Reader](#) role assignment or another Azure Resource Manager role assignment is necessary so that the user

can view and navigate storage account management resources in the Azure portal. The RBAC roles that grant access to blob or queue data do not grant access to storage account management resources. To access blob or queue data in the portal, the user needs permissions to navigate storage account resources. For more information about this requirement, see [Assign the Reader role for portal access](#).

The built-in roles that support access to your blob or queue data include:

- **Storage Blob Data Owner**: For POSIX access control for Azure Data Lake Storage Gen2.
- **Storage Blob Data Contributor**: Read/write/delete permissions for blobs.
- **Storage Blob Data Reader**: Read-only permissions for blobs.
- **Storage Queue Data Contributor**: Read/write/delete permissions for queues.
- **Storage Queue Data Reader**: Read-only permissions for queues.

Custom roles can support different combinations of the same permissions provided by the built-in roles. For more information about creating custom RBAC roles, see [Custom roles for Azure resources](#) and [Understand role definitions for Azure resources](#).

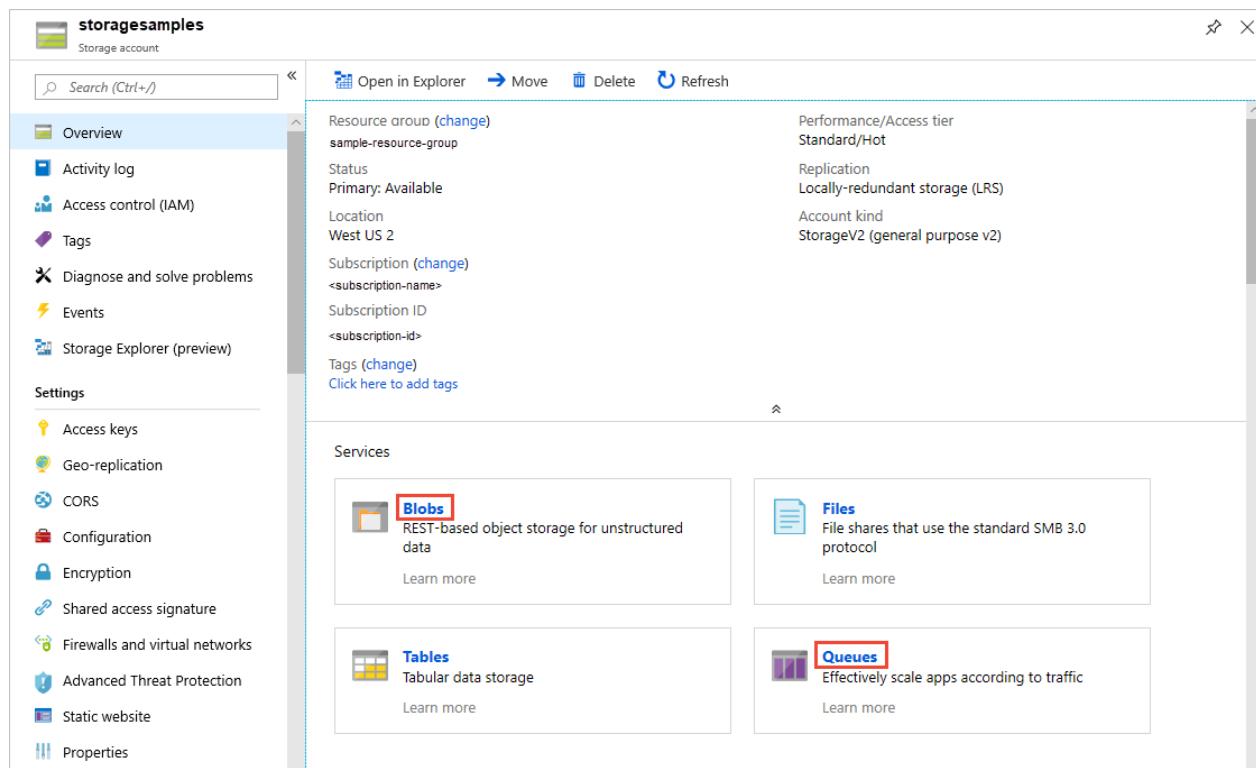
Listing queues with a classic subscription administrator role is not supported. To list queues, a user must have assigned to them the Azure Resource Manager **Reader** role, the **Storage Queue Data Reader** role, or the **Storage Queue Data Contributor** role.

#### IMPORTANT

The preview version of Storage Explorer in the Azure portal does not support using Azure AD credentials to view and modify blob or queue data. Storage Explorer in the Azure portal always uses the account keys to access data. To use Storage Explorer in the Azure portal, you must be assigned a role that includes `Microsoft.Storage/storageAccounts/listkeys/action`.

## Navigate to blobs or queues in the portal

To view blob or queue data in the portal, navigate to the **Overview** for your storage account, and click on the links for **Blobs** or **Queues**. Alternatively you can navigate to the **Blob service** and **Queue service** sections in the menu.



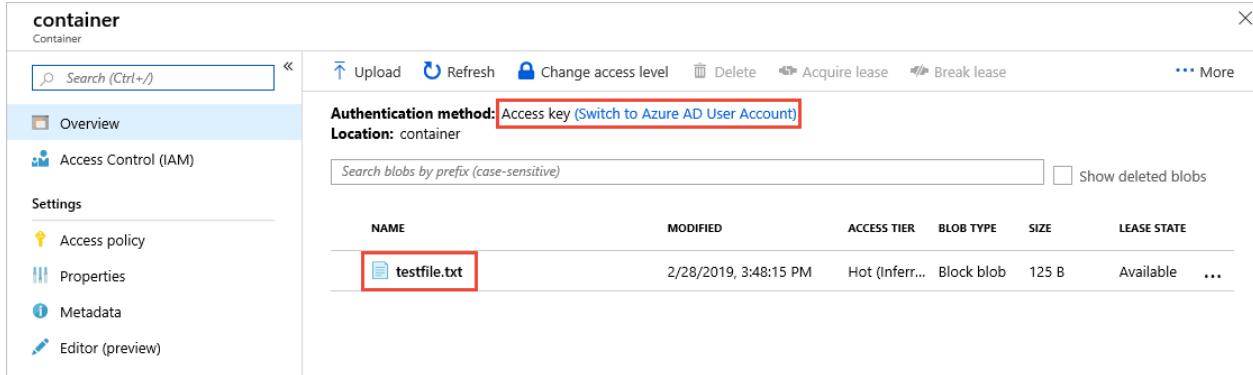
# Determine the current authentication method

When you navigate to a container or a queue, the Azure portal indicates whether you are currently using the account access key or your Azure AD account to authenticate.

The examples in this section show accessing a container and its blobs, but the portal displays the same message when you are accessing a queue and its messages, or listing queues.

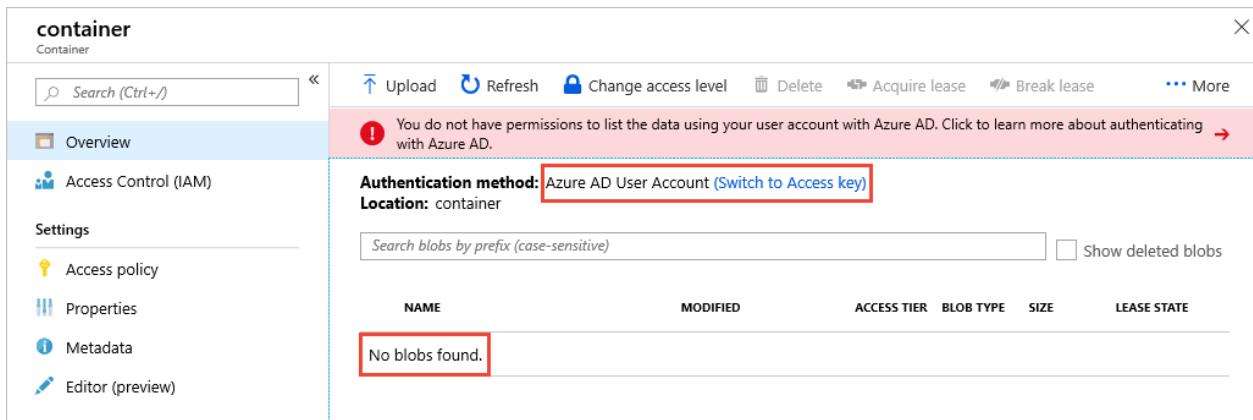
## Authenticate with the account access key

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 portal interface for a container named "container". On the left, there's a sidebar with options like Overview, Access Control (IAM), Settings (Access policy, Properties, Metadata, Editor (preview)), and a search bar. The main area has a toolbar with Upload, Refresh, Change access level, Delete, Acquire lease, Break lease, and More. Below the toolbar, it says "Authentication method: Access key (Switch to Azure AD User Account)" and "Location: container". A search bar for blobs is present. The blob list table has columns: NAME, MODIFIED, ACCESS TIER, BLOB TYPE, SIZE, and LEASE STATE. One row is shown: "testfile.txt" (MODIFIED: 2/28/2019, 3:48:15 PM, ACCESS TIER: Hot (Inferred), BLOB TYPE: Block blob, SIZE: 125 B, LEASE STATE: Available). The entire "Access key" text and the "testfile.txt" row are highlighted with red boxes.

To switch to using Azure AD account, click the link highlighted in the image. If you have the appropriate permissions via the RBAC 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:

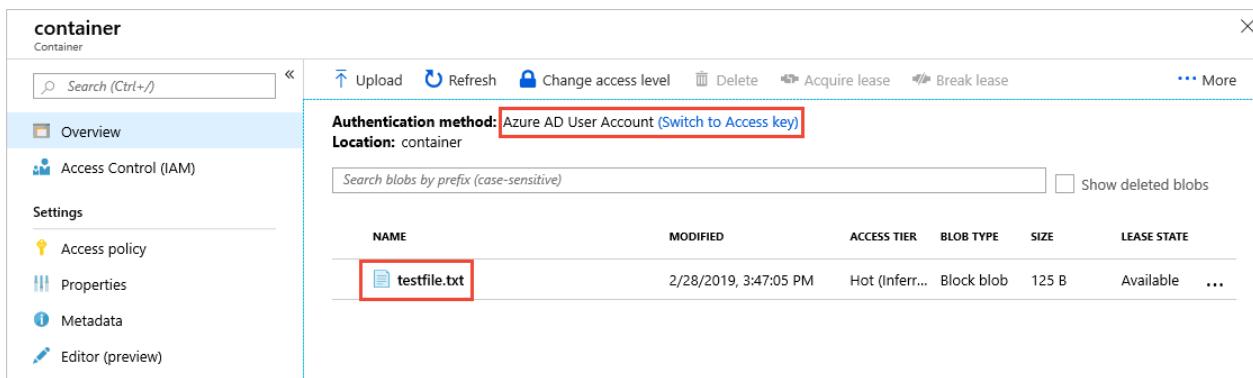


The screenshot shows the Azure portal interface for a container named "container". The sidebar and toolbar are similar to the previous screenshot. The main area shows an error message: "You do not have permissions to list the data using your user account with Azure AD. Click to learn more about authenticating with Azure AD." Below this, it says "Authentication method: Azure AD User Account (Switch to Access key)" and "Location: container". The search bar and blob list table are present. The entire "Azure AD User Account" text and the "No blobs found." message are highlighted with red boxes.

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 Azure portal interface for a container named "container". The sidebar and toolbar are similar to the previous screenshots. The main area shows the "Authentication method: Azure AD User Account (Switch to Access key)" and "Location: container" message. The search bar and blob list table are present. The entire "Azure AD User Account" text and the "testfile.txt" row are highlighted with red boxes.

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, and Editor (preview). The main content area has a search bar and buttons for Upload, Refresh, Change access level, Delete, Acquire lease, Break lease, and More. A prominent red warning message at the top states: 'You do not have permissions to use the access key to list data. Click to learn more about authenticating with Azure Storage.' Below this, under 'Authentication method:', it says 'Access key (Switch to Azure AD User Account)'. The 'Location' is listed as 'container'. A table titled 'No blobs found.' is shown with columns: NAME, MODIFIED, ACCESS TIER, BLOB TYPE, SIZE, and LEASE STATE. The entire 'No blobs found.' row is highlighted with a red box.

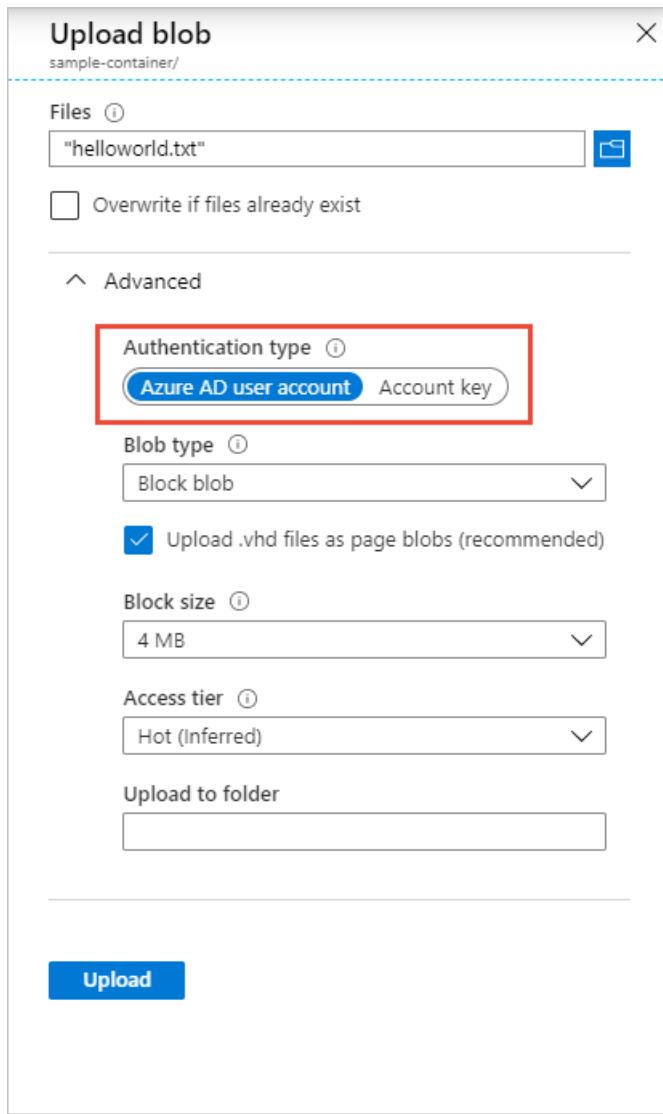
Notice that no blobs appear in the list if you do not have access to the account keys. Click on the [Switch to Azure AD User Account](#) link to use your Azure AD account for authentication again.

## 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:



## Next steps

- [Authenticate access to Azure blobs and queues using Azure Active Directory](#)
- [Grant access to Azure containers and queues with RBAC in the Azure portal](#)
- [Grant access to Azure blob and queue data with RBAC using Azure CLI](#)
- [Grant access to Azure blob and queue data with RBAC using PowerShell](#)

# Run PowerShell commands with Azure AD credentials to access blob or queue data

12/30/2019 • 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 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 and queue data to an Azure AD security principal via role-based access control (RBAC). For more information about RBAC roles in Azure Storage, see [Manage access rights to Azure Storage data with RBAC](#).

## Supported operations

The Azure Storage extensions are supported for operations on blob and queue 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 or queues are assigned via 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 or queue. If you have been assigned the **Blob Data Contributor** role, then you can run scripting commands that read, write, or delete a container or queue or the data they contain.

For details about the permissions required for each Azure Storage operation on a container or queue, see [Call storage operations with OAuth tokens](#).

## Call PowerShell commands using Azure AD credentials

### NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

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"
$location = "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 RBAC roles, see [Grant access to Azure blob and queue data with RBAC in the Azure portal](#).

**IMPORTANT**

RBAC 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

- [Use PowerShell to assign an RBAC role for access to blob and queue data](#)
- [Authorize access to blob and queue data with managed identities for Azure resources](#)

2 minutes to read

# Use the Azure portal to assign an RBAC role for access to blob and queue data

4/1/2020 • 8 minutes to read • [Edit Online](#)

Azure Active Directory (Azure AD) authorizes access rights to secured resources through [role-based access control \(RBAC\)](#). Azure Storage defines a set of built-in RBAC roles that encompass common sets of permissions used to access blob or queue data.

When an RBAC role is assigned to an Azure AD security principal, Azure grants access to those resources for that security principal. Access can be scoped to the level of the subscription, the resource group, the storage account, or an individual container or queue. An Azure AD security principal may be a user, a group, an application service principal, or a [managed identity for Azure resources](#).

This article describes how to use the Azure portal to assign RBAC roles. The Azure portal provides a simple interface for assigning RBAC roles and managing access to your storage resources. You can also assign RBAC roles for blob and queue resources using Azure command-line tools or the Azure Storage management APIs. For more information about RBAC roles for storage resources, see [Authenticate access to Azure blobs and queues using Azure Active Directory](#).

## RBAC roles for blobs and queues

Azure provides the following built-in RBAC roles for authorizing access to blob and queue data using Azure AD and OAuth:

- [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 Queue Data Contributor](#): Use to grant read/write/delete permissions to Azure queues.
- [Storage Queue Data Reader](#): Use to grant read-only permissions to Azure queues.
- [Storage Queue Data Message Processor](#): Use to grant peek, retrieve, and delete permissions to messages in Azure Storage queues.
- [Storage Queue Data Message Sender](#): Use to grant add permissions to messages in Azure Storage queues.

For detailed information about built-in RBAC 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 RBAC roles](#), and [Azure AD roles](#).

### NOTE

RBAC role assignments may take up to five minutes to propagate.

Only roles explicitly defined for data access permit a security principal to access blob or queue data. 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 or queue data within that account.

Access to blob or queue data in the Azure portal can be authorized using either your Azure AD account or the storage account access key. For more information, see [Use the Azure portal to access blob or queue data](#).

## Determine resource scope

Before you assign an 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.

The following list describes the levels at which you can scope access to Azure blob and queue resources, starting 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.
- **An individual queue.** At this scope, a role assignment applies to messages in the queue, as well as queue properties and metadata.
- **The storage account.** At this scope, a role assignment applies to all containers and their blobs, or to all queues and their messages.
- **The resource group.** At this scope, a role assignment applies to all of the containers or queues in all of the storage accounts in the resource group.
- **The subscription.** At this scope, a role assignment applies to all of the containers or queues in all of the storage accounts in all of the resource groups in the subscription.

### IMPORTANT

If your subscription includes an Azure DataBricks namespace, roles that are scoped to the subscription will not grant access to blob and queue data. Scope roles to the resource group, storage account, or container or queue instead.

## Assign RBAC roles using the Azure portal

After you have determined the appropriate scope for a role assignment, navigate to that resource in the Azure portal. Display the **Access Control (IAM)** settings for the resource, and follow these instructions to manage role assignments:

1. Assign the appropriate Azure Storage RBAC role to grant access to an Azure AD security principal.
2. Assign the Azure Resource Manager **Reader** role to users who need to access containers or queues via the Azure portal using their Azure AD credentials.

The following sections describe each of these steps in more detail.

### NOTE

As an owner of your Azure Storage account, you are not automatically assigned permissions to access data. You must explicitly assign yourself an RBAC role for Azure Storage. You can assign it at the level of your subscription, resource group, storage account, or a container or queue.

You cannot assign a role scoped to a container or queue if your storage account has a hierarchical namespace enabled.

### Assign a built-in RBAC role

Before you assign a role to a security principal, be sure to consider the scope of the permissions you are granting. Review the [Determine resource scope](#) section to decide the appropriate scope.

The procedure shown here assigns a role scoped to a container, but you can follow the same steps to assign a role scoped to a queue:

1. In the [Azure portal](#), go to your storage account and display the **Overview** for the account.
2. Under Services, select **Blobs**.

- Locate the container for which you want to assign a role, and display the container's settings.
- Select **Access control (IAM)** to display access control settings for the container. Select the **Role assignments** tab to see the list of role assignments.

The screenshot shows the Microsoft Azure portal interface. The URL in the address bar is <https://portal.azure.com/#@contoso.onmicrosoft.com/resource/suscriptions>. The top navigation bar includes links for Home, Storage accounts, sample-container - Access control (IAM), and Admin@contoso.com CONTOSO. The main content area is titled "sample-container - Access control (IAM)" and shows the "Role assignments" tab selected. It displays a table of 17 items (12 Users, 5 Service Principals) with columns for Name, Type, Role, and Scope. The table lists three users: AD Admin, AZ Admin 2, and RL Robert, all assigned the Contributor role at the Subscription level. The left sidebar has a tree view with "Access control (IAM)" highlighted by a red box.

- Click the **Add role assignment** button to add a new role.
- In the **Add role assignment** window, select the Azure Storage role that you want to assign. Then search to locate the security principal to which you want to assign that role.

The screenshot shows the "Add role assignment" dialog box. The "Role" dropdown is set to "Storage Blob Data Reader". The "Assign access to" dropdown is set to "Azure AD user, group, or service principal". The "Select" dropdown shows "azure-user" selected. Below, the "Selected members:" section shows "az" (with a green checkmark) and "azure-user". At the bottom are "Save" and "Discard" buttons.

- Click **Save**. The identity to whom you assigned the role appears listed under that role. For example, the following image shows that the user added now has read permissions to data in the container named *sample-container*.

The screenshot shows the Azure portal's 'Role assignments' section for a storage account. It lists two entries: 'App2' (App) with the 'Reader' role and 'Subscription (Inherited)', and 'STORAGE BLOB DATA READER (PREVIEW)' (User) with the 'Storage Blob Data Reader ... This resource' role assigned to 'azure-user'. The 'STORAGE BLOB DATA READER (PREVIEW)' entry is highlighted with a red border.

You can follow similar steps to assign a role scoped to the storage account, resource group, or subscription.

### Assign the Reader role for portal access

When you assign a built-in or custom role for Azure Storage to a security principal, you are granting permissions to that security principal to perform operations on data in your storage account. The built-in **Data Reader** roles provide read permissions for the data in a container or queue, while the built-in **Data Contributor** roles provide read, write, and delete permissions to a container or queue. Permissions are scoped to the specified resource.

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. Additional Azure AD permissions are required to navigate through the portal and view the other resources that are visible there.

If your users need to be able to access blobs in the Azure portal, then assign them an additional RBAC role, the **Reader** role, to those users, at the level of the storage account or above. 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.

Follow these steps to assign the **Reader** role so that a user can access blobs from the Azure portal. In this example, the assignment is scoped to the storage account:

1. In the [Azure portal](#), navigate to your storage account.
2. Select **Access control (IAM)** to display the access control settings for the storage account. Select the **Role assignments** tab to see the list of role assignments.
3. In the **Add role assignment** window, select the **Reader** role.
4. From the **Assign access to** field, select **Azure AD user, group, or service principal**.
5. Search to locate the security principal to which you want to assign the role.
6. Save the role assignment.

Assigning the **Reader** role is necessary only for users who need to access blobs or queues using the Azure portal.

#### IMPORTANT

The preview version of Storage Explorer in the Azure portal does not support using Azure AD credentials to view and modify blob or queue data. Storage Explorer in the Azure portal always uses the account keys to access data. To use Storage Explorer in the Azure portal, you must be assigned a role that includes **Microsoft.Storage/storageAccounts/listkeys/action**.

## Next steps

- For more information about RBAC roles for storage resources, see [Authenticate access to Azure blobs and queues using Azure Active Directory](#).
- To learn more about RBAC, see [What is role-based access control \(RBAC\)?](#).
- To learn how to assign and manage RBAC role assignments with Azure PowerShell, Azure CLI, or the REST API, see these articles:
  - [Manage role-based access control \(RBAC\) with Azure PowerShell](#)
  - [Manage role-based access control \(RBAC\) with Azure CLI](#)
  - [Manage role-based access control \(RBAC\) with the REST API](#)
- To learn how to authorize access to containers and queues from within your storage applications, see [Use Azure AD with Azure Storage applications](#).

# Use PowerShell to assign an RBAC role for access to blob and queue data

12/12/2019 • 6 minutes to read • [Edit Online](#)

Azure Active Directory (Azure AD) authorizes access rights to secured resources through [role-based access control \(RBAC\)](#). Azure Storage defines a set of built-in RBAC roles that encompass common sets of permissions used to access containers or queues.

When an RBAC role is assigned to an Azure AD security principal, Azure grants access to those resources for that security principal. Access can be scoped to the level of the subscription, the resource group, the storage account, or an individual container or queue. An Azure AD security principal may be a user, a group, an application service principal, or a [managed identity for Azure resources](#).

This article describes how to use Azure PowerShell to list built-in RBAC roles and assign them to users. For more information about using Azure PowerShell, see [Overview of Azure PowerShell](#).

## NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

## RBAC roles for blobs and queues

Azure provides the following built-in RBAC roles for authorizing access to blob and queue data using Azure AD and OAuth:

- [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 Queue Data Contributor](#): Use to grant read/write/delete permissions to Azure queues.
- [Storage Queue Data Reader](#): Use to grant read-only permissions to Azure queues.
- [Storage Queue Data Message Processor](#): Use to grant peek, retrieve, and delete permissions to messages in Azure Storage queues.
- [Storage Queue Data Message Sender](#): Use to grant add permissions to messages in Azure Storage queues.

For detailed information about built-in RBAC 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 RBAC roles](#), and [Azure AD roles](#).

#### NOTE

RBAC role assignments may take up to five minutes to propagate.

Only roles explicitly defined for data access permit a security principal to access blob or queue data. 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 or queue data within that account.

Access to blob or queue data in the Azure portal can be authorized using either your Azure AD account or the storage account access key. For more information, see [Use the Azure portal to access blob or queue data](#).

## Determine resource scope

Before you assign an 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.

The following list describes the levels at which you can scope access to Azure blob and queue resources, starting 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.
- **An individual queue.** At this scope, a role assignment applies to messages in the queue, as well as queue properties and metadata.
- **The storage account.** At this scope, a role assignment applies to all containers and their blobs, or to all queues and their messages.
- **The resource group.** At this scope, a role assignment applies to all of the containers or queues in all of the storage accounts in the resource group.
- **The subscription.** At this scope, a role assignment applies to all of the containers or queues in all of the storage accounts in all of the resource groups in the subscription.

#### IMPORTANT

If your subscription includes an Azure DataBricks namespace, roles that are scoped to the subscription will not grant access to blob and queue data. Scope roles to the resource group, storage account, or container or queue instead.

## List available RBAC roles

To list available built-in RBAC roles with Azure PowerShell, use the [Get-AzRoleDefinition](#) command:

```
Get-AzRoleDefinition | FT Name, Description
```

You'll see the built-in Azure Storage data roles listed, together with other built-in roles for Azure:

Storage Blob Data Contributor	Allows for read, write and delete access to Azure Storage blob containers and data
Storage Blob Data Owner	Allows for full access to Azure Storage blob containers and data, including assigning POSIX access control.
Storage Blob Data Reader	Allows for read access to Azure Storage blob containers and data
Storage Queue Data Contributor	Allows for read, write, and delete access to Azure Storage queues and queue messages
Storage Queue Data Message Processor	Allows for peek, receive, and delete access to Azure Storage queue messages
Storage Queue Data Message Sender	Allows for sending of Azure Storage queue messages
Storage Queue Data Reader	Allows for read access to Azure Storage queues and queue messages

# Assign an RBAC role to a security principal

To assign an RBAC role to a security principal, use the [New-AzRoleAssignment](#) command. The format of the command can differ based on the scope of the assignment. In order to run the command, you need to have Owner or Contributor role assigned at the corresponding scope. The following examples show how to assign a role to a user at various scopes, but you can use the same command to assign a role to any security principal.

## Container scope

To assign a role scoped to a container, specify a string containing the scope of the container for the `--scope` parameter. The scope for a container is in the form:

```
/subscriptions/<subscription>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>/blobServices/default/containers/<container-name>
```

The following example assigns the **Storage Blob Data Contributor** role to a user, scoped to a container named *sample-container*. Make sure to replace the sample values and the placeholder values in brackets with your own values:

```
New-AzRoleAssignment -SignInName <email> `
 -RoleDefinitionName "Storage Blob Data Contributor" `
 -Scope "/subscriptions/<subscription>/resourceGroups/sample-resource-group/providers/Microsoft.Storage/storageAccounts/<storage-account>/blobServices/default/containers/sample-container"
```

## Queue scope

To assign a role scoped to a queue, specify a string containing the scope of the queue for the `--scope` parameter. The scope for a queue is in the form:

```
/subscriptions/<subscription>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>/queueServices/default/queues/<queue-name>
```

The following example assigns the **Storage Queue Data Contributor** role to a user, scoped to a queue named *sample-queue*. Make sure to replace the sample values and the placeholder values in brackets with your own values:

```
New-AzRoleAssignment -SignInName <email> `
 -RoleDefinitionName "Storage Queue Data Contributor" `
 -Scope "/subscriptions/<subscription>/resourceGroups/sample-resource-group/providers/Microsoft.Storage/storageAccounts/<storage-account>/queueServices/default/queues/sample-queue"
```

## Storage account scope

To assign a role scoped to the storage account, specify the scope of the storage account resource for the `--scope` parameter. The scope for a storage account is in the form:

```
/subscriptions/<subscription>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>
```

The following example shows how to scope the **Storage Blob Data Reader** role to a user at the level of the storage account. Make sure to replace the sample values with your own values:

```
New-AzRoleAssignment -SignInName <email> `
 -RoleDefinitionName "Storage Blob Data Reader" `
 -Scope "/subscriptions/<subscription>/resourceGroups/sample-resource-
group/providers/Microsoft.Storage/storageAccounts/<storage-account>"
```

## Resource group scope

To assign a role scoped to the resource group, specify the resource group name or ID for the `--resource-group` parameter. The following example assigns the **Storage Queue Data Reader** role to a user at the level of the resource group. Make sure to replace the sample values and placeholder values in brackets with your own values:

```
New-AzRoleAssignment -SignInName <email> `
 -RoleDefinitionName "Storage Queue Data Reader" `
 -ResourceGroupName "sample-resource-group"
```

## Subscription scope

To assign a role scoped to the subscription, specify the scope for the subscription for the `--scope` parameter. The scope for a subscription is in the form:

```
/subscriptions/<subscription>
```

The following example shows how to assign the **Storage Blob Data Reader** role to a user at the level of the storage account. Make sure to replace the sample values with your own values:

```
New-AzRoleAssignment -SignInName <email> `
 -RoleDefinitionName "Storage Blob Data Reader" `
 -Scope "/subscriptions/<subscription>"
```

## Next steps

- [Manage access to Azure resources using RBAC and Azure PowerShell](#)
- [Grant access to Azure blob and queue data with RBAC using Azure CLI](#)
- [Grant access to Azure blob and queue data with RBAC in the Azure portal](#)

# Use Azure CLI to assign an RBAC role for access to blob and queue data

12/5/2019 • 6 minutes to read • [Edit Online](#)

Azure Active Directory (Azure AD) authorizes access rights to secured resources through [role-based access control \(RBAC\)](#). Azure Storage defines a set of built-in RBAC roles that encompass common sets of permissions used to access blob or queue data.

When an RBAC role is assigned to an Azure AD security principal, Azure grants access to those resources for that security principal. Access can be scoped to the level of the subscription, the resource group, the storage account, or an individual container or queue. An Azure AD security principal may be a user, a group, an application service principal, or a [managed identity for Azure resources](#).

This article describes how to use Azure CLI to list built-in RBAC roles and assign them to users. For more information about using Azure CLI, see [Azure Command-Line Interface \(CLI\)](#).

## RBAC roles for blobs and queues

Azure provides the following built-in RBAC roles for authorizing access to blob and queue data using Azure AD and OAuth:

- [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 Queue Data Contributor](#): Use to grant read/write/delete permissions to Azure queues.
- [Storage Queue Data Reader](#): Use to grant read-only permissions to Azure queues.
- [Storage Queue Data Message Processor](#): Use to grant peek, retrieve, and delete permissions to messages in Azure Storage queues.
- [Storage Queue Data Message Sender](#): Use to grant add permissions to messages in Azure Storage queues.

For detailed information about built-in RBAC 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 RBAC roles](#), and [Azure AD roles](#).

### NOTE

RBAC role assignments may take up to five minutes to propagate.

Only roles explicitly defined for data access permit a security principal to access blob or queue data. 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 or queue data within that account.

Access to blob or queue data in the Azure portal can be authorized using either your Azure AD account or the storage account access key. For more information, see [Use the Azure portal to access blob or queue data](#).

## Determine resource scope

Before you assign an 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.

The following list describes the levels at which you can scope access to Azure blob and queue resources, starting 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.
- **An individual queue.** At this scope, a role assignment applies to messages in the queue, as well as queue properties and metadata.
- **The storage account.** At this scope, a role assignment applies to all containers and their blobs, or to all queues and their messages.
- **The resource group.** At this scope, a role assignment applies to all of the containers or queues in all of the storage accounts in the resource group.
- **The subscription.** At this scope, a role assignment applies to all of the containers or queues in all of the storage accounts in all of the resource groups in the subscription.

#### IMPORTANT

If your subscription includes an Azure DataBricks namespace, roles that are scoped to the subscription will not grant access to blob and queue data. Scope roles to the resource group, storage account, or container or queue instead.

## List available RBAC roles

To list available built-in RBAC roles with Azure CLI, use the [az role definition list](#) command:

```
az role definition list --out table
```

You'll see the built-in Azure Storage data roles listed, together with other built-in roles for Azure:

Storage Blob Data Contributor	Allows for read, write and delete access to Azure Storage blob containers and data
Storage Blob Data Owner	Allows for full access to Azure Storage blob containers and data, including assigning POSIX access control.
Storage Blob Data Reader	Allows for read access to Azure Storage blob containers and data
Storage Queue Data Contributor	Allows for read, write, and delete access to Azure Storage queues and queue messages
Storage Queue Data Message Processor	Allows for peek, receive, and delete access to Azure Storage queue messages
Storage Queue Data Message Sender	Allows for sending of Azure Storage queue messages
Storage Queue Data Reader	Allows for read access to Azure Storage queues and queue messages

## Assign an RBAC role to a security principal

To assign an RBAC role to a security principal, use the [az role assignment create](#) command. The format of the command can differ based on the scope of the assignment. The following examples show how to assign a role to a user at various scopes, but you can use the same command to assign a role to any security principal.

### Container scope

To assign a role scoped to a container, specify a string containing the scope of the container for the `--scope` parameter. The scope for a container is in the form:

```
/subscriptions/<subscription>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>/blobServices/default/containers/<container>
```

The following example assigns the **Storage Blob Data Contributor** role to a user, scoped to the level of the container. Make sure to replace the sample values and the placeholder values in 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>/blobServices/default/containers/<container>"
```

## Queue scope

To assign a role scoped to a queue, specify a string containing the scope of the queue for the `--scope` parameter. The scope for a queue is in the form:

```
/subscriptions/<subscription>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>/queueServices/default/queues/<queue>
```

The following example assigns the **Storage Queue Data Contributor** role to a user, scoped to the level of the queue. Make sure to replace the sample values and the placeholder values in brackets with your own values:

```
az role assignment create \
--role "Storage Queue Data Contributor" \
--assignee <email> \
--scope "/subscriptions/<subscription>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>/queueServices/default/queues/<queue>"
```

## Storage account scope

To assign a role scoped to the storage account, specify the scope of the storage account resource for the `--scope` parameter. The scope for a storage account is in the form:

```
/subscriptions/<subscription>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>
```

The following example shows how to assign the **Storage Blob Data Reader** role to a user at the level of the storage account. Make sure to replace the sample values with your own values:

```
az role assignment create \
--role "Storage Blob Data Reader" \
--assignee <email> \
--scope "/subscriptions/<subscription>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>"
```

## Resource group scope

To assign a role scoped to the resource group, specify the resource group name or ID for the `--resource-group` parameter. The following example assigns the **Storage Queue Data Reader** role to a user at the level of the resource group. Make sure to replace the sample values and placeholder values in brackets with your own values:

```
az role assignment create \
--role "Storage Queue Data Reader" \
--assignee <email> \
--resource-group <resource-group>
```

## Subscription scope

To assign a role scoped to the subscription, specify the scope for the subscription for the `--scope` parameter. The scope for a subscription is in the form:

```
/subscriptions/<subscription>
```

The following example shows how to assign the **Storage Blob Data Reader** role to a user at the level of the storage account. Make sure to replace the sample values with your own values:

```
az role assignment create \
--role "Storage Blob Data Reader" \
--assignee <email> \
--scope "/subscriptions/<subscription>"
```

## Next steps

- [Manage access to Azure resources using RBAC and Azure PowerShell](#)
- [Grant access to Azure blob and queue data with RBAC using Azure PowerShell](#)
- [Grant access to Azure blob and queue data with RBAC in the Azure portal](#)

# Authorize access to blob and queue data with managed identities for Azure resources

1/14/2020 • 6 minutes to read • [Edit Online](#)

Azure Blob and Queue storage support Azure Active Directory (Azure AD) authentication with [managed identities for Azure resources](#). Managed identities for Azure resources can authorize access to blob and queue 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 or queue data from an Azure VM using managed identities for Azure Resources. It also describes how to test your code in the development environment.

## Enable managed identities on a VM

Before you can use managed identities for Azure Resources to authorize access to blobs and queues 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](#).

## Authenticate with the Azure Identity library

The Azure Identity client library provides Azure AD token authentication support for the [Azure SDK](#). The latest versions of the Azure Storage client libraries for .NET, Java, Python, and JavaScript integrate with the Azure Identity library to provide a simple and secure means to acquire an OAuth 2.0 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 authenticate whether your application is running in the development environment or in Azure. The Azure Identity client library for .NET authenticates a security principal. When your code is running in Azure, the security principal is a managed identity for Azure resources. In the development environment, the managed identity does not exist, so the client library authenticates either the user or a service principal for testing purposes.

After authenticating, the Azure Identity client library gets a token credential. This token credential is then encapsulated in the service client object that you create to perform operations against Azure Storage. The library handles this for you seamlessly by getting the appropriate token credential.

For more information about the Azure Identity client library for .NET, see [Azure Identity client library for .NET](#). For reference documentation for the Azure Identity client library, see [Azure.Identity Namespace](#).

### Assign role-based access control (RBAC) roles for access to data

When an Azure AD security principal attempts to access blob or queue 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 RBAC role that grants access to blob or queue data in Azure Storage. For information about assigning permissions via RBAC, see the section titled **Assign RBAC roles for access rights** in [Authorize access to Azure blobs and queues using Azure Active Directory](#).

## 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 supports 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](#).

Other development tools may prompt you to login via a web browser.

## 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.

### Create the service principal

To create a service principal with Azure CLI and assign an RBAC 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 [Built-in roles for Azure resources](#).

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 Reader** role to it with account scope

```
az ad sp create-for-rbac \
--name <service-principal> \
--role "Storage Blob Data Reader" \
--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"
}
```

### IMPORTANT

RBAC role assignments may take a few 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](#).

## 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
```

## .NET code example: Create a block blob

Add the following `using` directives to your code to use the Azure Identity and Azure Storage client libraries.

```
using Azure;
using Azure.Identity;
using Azure.Storage.Blobs;
using System;
using System.IO;
using System.Text;
using System.Threading.Tasks;
```

To get a token credential that your code can use to authorize requests to Azure Storage, create an instance of the [DefaultAzureCredential](#) class. The following code example shows how to get the authenticated token credential and use it to create a service client object, then use 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://'{0}'.blob.core.windows.net/{1}",
 accountName,
 containerName);

 // Get a credential and create a 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 or queue data with Azure AD, you must use HTTPS for those requests.

## Next steps

- [Manage access rights to storage data with RBAC.](#)
- [Use Azure AD with storage applications.](#)
- [Run Azure CLI or PowerShell commands with Azure AD credentials to access blob or queue data.](#)

# Acquire a token from Azure AD for authorizing requests from a client application

2/12/2020 • 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 (formerly Azure AD). 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 Microsoft identity platform 2.0. The code example features .NET, but other languages use a similar approach. For more information about Microsoft identity platform 2.0, see [Microsoft identity platform \(v2.0\) 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](#).

## Assign a role to an Azure AD security principal

To authenticate a security principal from your Azure Storage application, first configure role-based access control (RBAC) settings for that security principal. Azure Storage defines built-in RBAC roles that encompass permissions for containers and queues. When the RBAC role is assigned to a security principal, that security principal is granted access to that resource. For more information, see [Manage access rights to Azure Blob and Queue data with RBAC](#).

## 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:

Dashboard > Microsoft - App registrations > Register an application

## Register an application

**⚠** If you are building an application for external users that will be distributed by Microsoft, you must register as a first party application to meet all security, privacy, and compliance policies. [Read our decision guide](#)

**\* 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  
 Accounts in any organizational directory  
 Accounts in any organizational directory and personal Microsoft accounts (e.g. Skype, Xbox, Outlook.com)  
[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.  
  ✓

By proceeding, you agree to the Microsoft Platform Policies [\[link\]](#)

**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

### StorageClientAppSample

Overview	Display name StorageClientAppSample	Supported account types My organization only
Quickstart	Application (client) ID <application-id>	Redirect URIs 1 web, 0 public client
Manage	Directory (tenant) ID <directory-id>	Managed application in local directory <a href="#">Create Service Principal</a>
Branding	Object ID <object-id>	
Authentication		

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 **Overview** page for your registered application, select **View API Permissions**.
2. In the **API permissions** section, select **Add a permission** and choose **Microsoft APIs**.
3. Select **Azure Storage** from the list of results to display the **Request API permissions** pane.
4. 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.
5. In the **Select permissions** section of the **Request API permissions** pane, select the checkbox next to **user\_impersonation**, then click **Add permissions**.

The screenshot shows the 'Request API permissions' pane. At the top, it says 'Azure Storage'. Below that, it asks 'What type of permissions does your application require?'. Two options are shown: 'Delegated permissions' (selected) and 'Application permissions'. Under 'Delegated permissions', it says 'Your application needs to access the API as the signed-in user.' Under 'Application permissions', it says 'Your application runs as a background service or daemon without a signed-in user.'. Below this, there's a table titled 'Select permissions' with columns 'PERMISSION' and 'ADMIN CONSENT REQUIRED'. A row for 'user\_impersonation' (Access Azure Storage) has a checked checkbox. At the bottom are 'Add permissions' and 'Discard' buttons.

The **API permissions** pane now shows that your registered Azure AD application has access to both Microsoft Graph and the Azure Storage. Permissions are granted to Microsoft Graph automatically when you first register your app with Azure AD.

The screenshot shows the 'API permissions' pane. It lists permissions for 'Azure Storage (1)' and 'Microsoft Graph (1)'. For 'Azure Storage (1)', there is one permission: 'user\_impersonation' (Delegated, Access Azure Storage). For 'Microsoft Graph (1)', there is one permission: 'User.Read' (Delegated, Sign in and read user profile). Both permissions have '-' under 'ADMIN CONSENT REQUIRED'. At the bottom, a note says: 'These are the permissions that this application requests statically. You may also request user consent-able permissions dynamically through code. See best practices for requesting permissions'.

## Create a client secret

The application needs a client secret to prove its identity when requesting a token. 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.

New client secret			
DESCRIPTION	EXPIRES	VALUE	
storage auth flow sample	6/6/2020	+oN*****	

## 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 content](#).

## 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
-------------	-------------

RESOURCE ID	DESCRIPTION
<code>https://&lt;account&gt;.blob.core.windows.net</code> <code>https://&lt;account&gt;.queue.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://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. For instructions on how to assign this role in the Azure portal, see [Grant access to Azure blob and queue data with RBAC in the Azure portal](#).

#### 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 RBAC role for Azure Storage. You can assign it at the level of your subscription, resource group, storage account, or container or queue.

### 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:

```
Install-Package Microsoft.Azure.Storage.Blob
Install-Package Microsoft.Azure.Storage.Common
```

Next, add the following using statements to the HomeController.cs file:

```
using Microsoft.Identity.Client; //MSAL library for getting the access token
using Microsoft.WindowsAzure.Storage.Auth;
using Microsoft.WindowsAzure.Storage.Blob;
```

### Create a block blob

Add the following code snippet to create a block blob:

```
private static async Task<string> CreateBlob(string accessToken)
{
 // Create a blob on behalf of the user
 TokenCredential tokenCredential = new TokenCredential(accessToken);
 StorageCredentials storageCredentials = new StorageCredentials(tokenCredential);

 // Replace the URL below with your storage account URL
 CloudBlockBlob blob =
 new CloudBlockBlob(
 new Uri("https://<storage-account>.blob.core.windows.net/<container>/Blob1.txt"),
 storageCredentials);
 await blob.UploadTextAsync("Blob created by Azure AD authenticated user.");
 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 authorize the request 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 OAuth 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. When consent is necessary, the example catches the **MsalUiRequiredException** and calls another method to facilitate the request for consent:

```
public async Task<IActionResult> Blob()
{
 var scopes = new string[] { "https://storage.azure.com/user_impersonation" };
 try
 {
 var accessToken =
 await _tokenAcquisition.GetAccessTokenOnBehalfOfUser(HttpContext, scopes);
 ViewData["Message"] = await CreateBlob(accessToken);
 return View();
 }
 catch (MsalUiRequiredException ex)
 {
 AuthenticationProperties properties =
 BuildAuthenticationPropertiesForIncrementalConsent(scopes, ex);
 return Challenge(properties);
 }
}
```

Consent is the process of a user granting authorization to an application to access protected resources on their behalf. The Microsoft identity platform 2.0 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 at any given time by in the `scope` parameter. For more information about incremental consent, see the section titled **Incremental and dynamic consent** in [Why update to Microsoft identity platform \(v2.0\)?](#).

The following method constructs the authentication properties for requesting incremental consent:

```
private AuthenticationProperties BuildAuthenticationPropertiesForIncrementalConsent(string[] scopes,
 MsalUiRequiredException
ex)
{
 AuthenticationProperties properties = new AuthenticationProperties();

 // Set the scopes, including the scopes that ADAL.NET or MSAL.NET need for the Token cache.
 string[] additionalBuildInScopes = new string[] { "openid", "offline_access", "profile" };
 properties.SetParameter<ICollection<string>>(OpenIdConnectParameterNames.Scope,
 scopes.Union(additionalBuildInScopes).ToList());

 // Attempt to set the login_hint so that the logged-in user is not presented
 // with an account selection dialog.
 string loginHint = HttpContext.User.GetLoginHint();
 if (!string.IsNullOrWhiteSpace(loginHint))
 {
 properties.SetParameter<string>(OpenIdConnectParameterNames.LoginHint, loginHint);

 string domainHint = HttpContext.User.GetDomainHint();
 properties.SetParameter<string>(OpenIdConnectParameterNames.DomainHint, domainHint);
 }

 // Specify any additional claims that are required (for instance, MFA).
 if (!string.IsNullOrEmpty(ex.Claims))
 {
 properties.Items.Add("claims", ex.Claims);
 }

 return properties;
}
```

# 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

Next, 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>",
 "CallbackPath": "/signin-oidc",
 "SignedOutCallbackPath": "/signout-callback-oidc",

 // To call an API
 "ClientSecret": "<client-secret>"
 },
 "Logging": {
 "LogLevel": {
 "Default": "Warning"
 }
 },
 "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:

```
CloudBlockBlob blob = new CloudBlockBlob(
 new Uri("https://<storage-account>.blob.core.windows.net/<container>/Blob1.txt"),
 storageCredentials);
```

## Enable implicit grant flow

To run the sample, you may need to configure the implicit grant flow for your app registration. Follow these steps:

1. Navigate to your app registration in the Azure portal.
2. In the Manage section, select the **Authentication** setting.
3. Under **Advanced settings**, in the **Implicit grant** section, select the check boxes to enable access tokens and ID tokens, as shown in the following image:

## Advanced settings

Logout URL i e.g. <https://myapp.com/logout>

### Implicit grant

Allows an application to request a token directly from the authorization endpoint. Recommended only if the application has a single page architecture (SPA), has no backend components, or invokes a Web API via JavaScript.

To enable the implicit grant flow, select the tokens you would like to be issued by the authorization endpoint:

- Access tokens
- ID tokens

## Update the port used by localhost

When you run the sample, you may find that you need to update the redirect URI specified in your app registration to use the *localhost* port assigned at runtime. To update the redirect URI to use the assigned port, follow these steps:

1. Navigate to your app registration in the Azure portal.
2. In the Manage section, select the **Authentication** setting.
3. Under **Redirect URIs**, edit the port to match that used by the sample application, as shown in the following image:

### Redirect URIs

The URIs that we will accept as destinations when returning authentication responses (tokens) after successfully authenticating users. Also referred to as reply URLs.

[Learn more about adding support for web, mobile and desktop clients](#) i

TYPE	REDIRECT URI	
Web	<a href="http://localhost:63572/signin-oidc">http://localhost:63572/signin-oidc</a>	<span style="color: #0078d4;">i</span>
Web	<a href="https://myapp.com/auth">e.g. https://myapp.com/auth</a>	

## Next steps

- To learn more about the Microsoft identity platform, see [Microsoft identity platform](#).
- To learn more about RBAC roles for Azure storage, see [Manage access rights to storage data with RBAC](#).
- To learn about using managed identities for Azure resources with Azure Storage, see [Authenticate access to blobs and queues with Azure Active Directory and managed identities for Azure Resources](#).

# Manage storage account access keys

4/16/2020 • 4 minutes to read • [Edit Online](#)

When you create a storage account, Azure generates two 512-bit storage account access keys. 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.

If possible, use Azure Active Directory (Azure AD) to authorize requests to Blob and Queue storage instead of Shared Key. Azure AD provides superior security and ease of use over Shared Key. For more information about authorizing access to data with Azure AD, see [Authorize access to Azure blobs and queues using Azure Active Directory](#).

## View access keys and connection string

To view and copy your storage account access keys or connection string from the Azure portal:

1. Navigate to the [Azure portal](#).
2. Locate your storage account.
3. Under **Settings**, select **Access keys**. Your account access keys appear, as well as the complete connection string for each key.
4. Find the **Key** value under **key1**, and click the **Copy** button to copy the account key.
5. Alternately, you can copy the entire connection string. Find the **Connection string** value under **key1**, and click the **Copy** button to copy the connection string.



You can use either key 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 RBAC role that includes the **Microsoft.Storage/storageAccounts/listkeys/action**. Some built-in RBAC 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 RBAC 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](#).

Follow this process to rotate your storage account keys:

1. Update the connection strings in your application code to use the secondary key.
2. Regenerate the primary access key for your storage account. On the **Access Keys** blade in the Azure portal, click **Regenerate Key1**, and then click **Yes** to confirm that you want to generate a new key.
3. Update the connection strings in your code to reference the new primary access key.
4. Regenerate the secondary access key in the same manner.

### NOTE

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 RBAC role that includes the **Microsoft.Storage/storageAccounts/regeneratekey/action**. Some built-in RBAC 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 RBAC roles, and Azure AD roles](#). For detailed information about built-in RBAC roles for Azure Storage, see the **Storage** section in [Azure built-in roles for Azure RBAC](#).

## Next steps

- [Azure storage account overview](#)
- [Create a storage account](#)

# Configure Azure Storage connection strings

12/23/2019 • 9 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 Azure storage emulator.
- Access a storage account in Azure.
- Access specified resources in Azure via a shared access signature (SAS).

## Protect your access keys

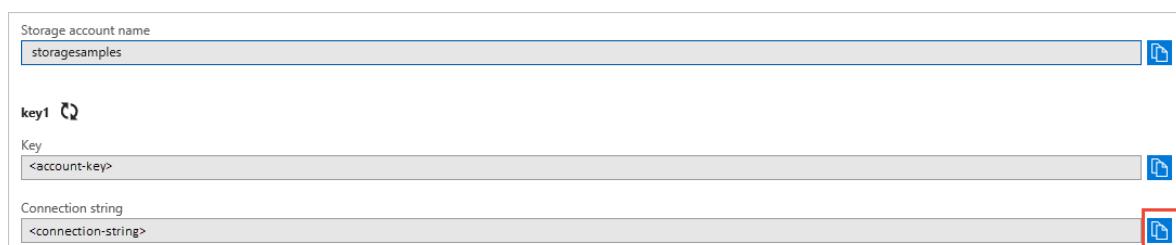
Your storage account access keys are similar to a root password for your storage account. Always be careful to protect your access keys. Use Azure Key Vault to manage and rotate your keys securely. Avoid distributing access keys to other users, hard-coding them, or saving them anywhere in plain text that is accessible to others. Rotate your keys if you believe they may have been compromised.

If possible, use Azure Active Directory (Azure AD) to authorize requests to Blob and Queue storage instead of Shared Key. Azure AD provides superior security and ease of use over Shared Key. For more information about authorizing access to data with Azure AD, see [Authorize access to Azure blobs and queues using Azure Active Directory](#).

## View and copy a connection string

To view and copy your storage account access keys or connection string from the Azure portal:

1. Navigate to the [Azure portal](#).
2. Locate your storage account.
3. Under **Settings**, select **Access keys**. Your account access keys appear, as well as the complete connection string for each key.
4. Find the **Key** value under **key1**, and click the **Copy** button to copy the account key.
5. Alternately, you can copy the entire connection string. Find the **Connection string** value under **key1**, and click the **Copy** button to copy the connection string.



You can use either key 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 RBAC role that includes the `Microsoft.Storage/storageAccounts/listkeys/action`. Some built-in RBAC 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 RBAC 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](#).

## 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 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 the storage emulator

The storage 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 storage emulator. They are:

```
Account name: devstoreaccount
Account key: Eby8vdM02xNOcqFlqUwJPLlEt1CDXJ10UzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==
```

### NOTE

The authentication key supported by the storage 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 storage emulator. You should not use the development account with production data.

The storage 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 a shortcut

The easiest way to connect to the storage emulator 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 to the storage emulator in an `app.config` file:

```
<appSettings>
 <add key="StorageConnectionString" value="UseDevelopmentStorage=true" />
</appSettings>
```

### Connect to the emulator account using the well-known account name and key

To create a connection string that references the emulator account name and key, you must specify the endpoints for each of the services you wish to use from the emulator in the connection string. This is necessary so that the connection string will reference the emulator endpoints, which are different than those for a production storage account. For example, the value of your connection string will look like this:

```
DefaultEndpointsProtocol=http;AccountName=devstoreaccount1;
AccountKey=Eby8vdM02xN0cqFlqUwJPl1mEt1CDXJ10UzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==;
BlobEndpoint=http://127.0.0.1:10000/devstoreaccount1;
TableEndpoint=http://127.0.0.1:10002/devstoreaccount1;
QueueEndpoint=http://127.0.0.1:10001/devstoreaccount1;
```

This value is identical to the shortcut shown above, `UseDevelopmentStorage=true`.

#### Specify an HTTP proxy

You can also specify an HTTP proxy to use when you're testing your service against the storage emulator. This can be useful for observing HTTP requests and responses while you're debugging operations against the storage services. To specify a proxy, add the `DevelopmentStorageProxyUri` option to the connection string, and set its value to the proxy URI. For example, here is a connection string that points to the storage emulator and configures an HTTP proxy:

```
UseDevelopmentStorage=true;DevelopmentStorageProxyUri=http://myProxyUri
```

For more information about the storage emulator, see [Use the Azure storage emulator for development and testing](#).

## 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.

#### IMPORTANT

Service endpoint values in your connection strings must be well-formed URIs, including `https://` (recommended) or `http://`. Because Azure Storage does not yet support HTTPS for custom domains, you *must* specify `http://` for any endpoint URI that points to a custom domain.

### 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 Azure storage emulator for development and testing](#)
- [Azure Storage explorers](#)
- [Using Shared Access Signatures \(SAS\)](#)

# Call REST API operations with Shared Key authorization

2/28/2020 • 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 `http://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:1dVlYJWWJAOSHTCPGiwdX1rOS8B4fenYP/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 for 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{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:

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\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](#)

# Determine which Azure Storage encryption key model is in use for the storage account

3/15/2020 • 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

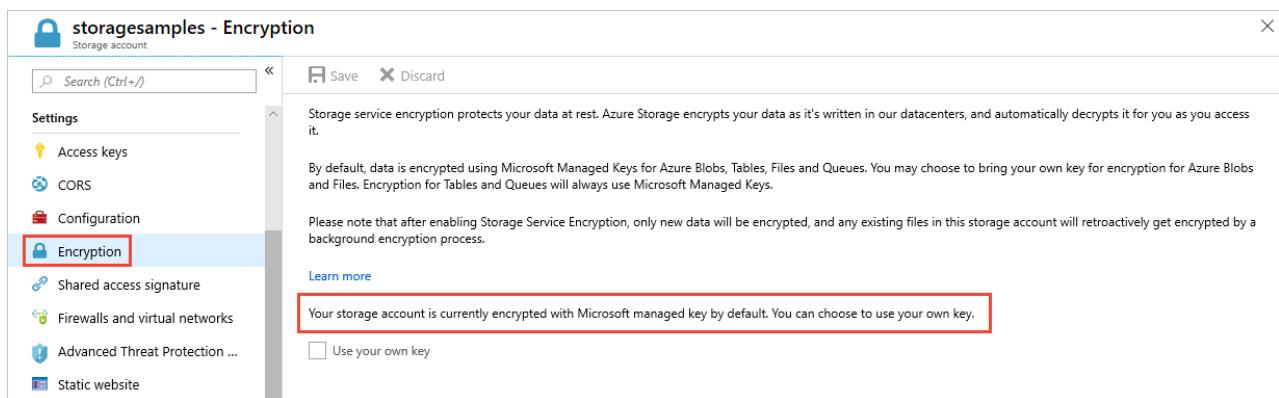
To determine whether a storage account is using Microsoft-managed keys or customer-managed keys for encryption, use one of the following approaches.

- [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:

The screenshot shows the 'Encryption' blade for the 'storagesamples' storage account. On the left is a navigation menu with links like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Data transfer, Events, and Storage Explorer (preview). The main area has a search bar and Save/Discard buttons. A section titled 'Use your own key' is highlighted with a red box. It contains an 'Encryption key' section with two options: 'Enter key URI' (selected) and 'Select from Key Vault'. Below is a 'Key URI' input field containing 'https://storage-key-vault-sample.vault.azure.net/keys/storage-sample-key/7e44c8e00fc42cdb1a84c63e4bd9ff6'. A note below the input field states: '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.' with a 'Learn more' link.

## Next steps

- [Azure Storage encryption for data at rest](#)
- [Use customer-managed keys with Azure Key Vault to manage Azure Storage encryption](#)

# Configure customer-managed keys with Azure Key Vault by using the Azure portal

4/16/2020 • 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 supply customer-managed keys to use for encryption of blob and file data.

Customer-managed keys must be stored in an Azure Key Vault. You can either create your own keys and store them in a key vault, or you can use the Azure Key Vault APIs to generate keys. The storage account and the key vault must be in the same region, but they can be in different subscriptions. For more information about Azure Storage encryption and key management, see [Azure Storage encryption for data at rest](#). For more information about Azure Key Vault, see [What is Azure Key Vault?](#)

This article shows how to configure an Azure Key Vault with customer-managed keys using the [Azure portal](#). To learn how to create a key vault using the Azure portal, see [Quickstart: Set and retrieve a secret from Azure Key Vault using the Azure portal](#).

## Configure Azure Key Vault

Using customer-managed keys with Azure Storage encryption requires that two properties be set on the key vault, **Soft Delete** and **Do Not Purge**. These properties are not enabled by default, but can be enabled using either PowerShell or Azure CLI on a new or existing key vault.

To learn how to enable these properties on an existing key vault, see the sections titled **Enabling soft-delete** and **Enabling Purge Protection** in one of the following articles:

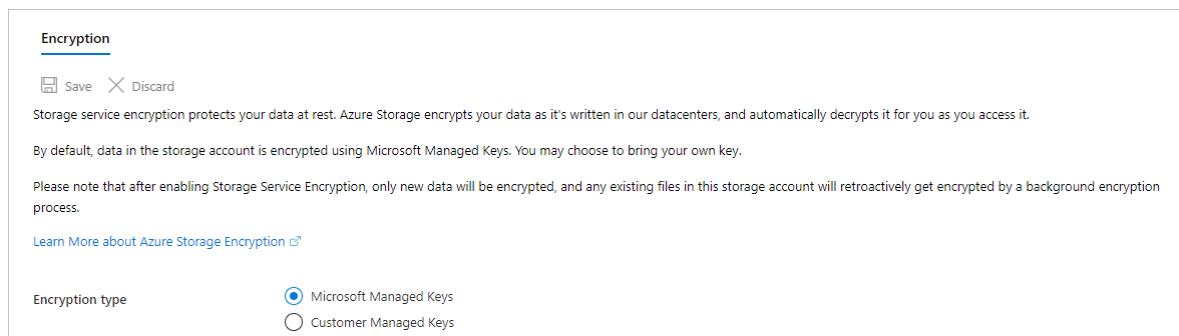
- [How to use soft-delete with PowerShell](#).
- [How to use soft-delete with CLI](#).

Only 2048-bit RSA and RSA-HSM keys are supported with Azure Storage encryption. For more information about keys, see [Key Vault keys](#) in [About Azure Key Vault keys, secrets and certificates](#).

## Enable customer-managed keys

To enable customer-managed keys in the Azure portal, follow these steps:

1. Navigate to your storage account.
2. On the **Settings** blade for the storage account, click **Encryption**. Select the **Customer Managed Keys** option, as shown in the following image.



# Specify a key

After you enable customer-managed keys, you'll have the opportunity to specify a key to associate with the storage account.

## Specify a key as a URI

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.

The screenshot shows the 'Keys' settings page in the Azure portal. A specific key version is selected, showing its properties: Key Type (RSA), RSA Key Size (2048), Created (4/9/2019, 12:50:38 PM), and Updated (4/9/2019, 12:50:38 PM). The 'Key Identifier' field contains the value '<key-uri>' with a copy icon. Below it, under 'Settings', there are options for activation and expiration dates, both of which are currently unchecked. The 'Enabled?' button is set to 'Yes'. Under 'Permitted operations', all six options (Encrypt, Decrypt, Sign, Verify, Wrap Key, Unwrap Key) are checked with blue checkmarks. There are also 'Tags' and a 'Permissions' section below.

3. In the **Encryption** settings for your storage account, choose the **Enter key URI** option.

4. Paste the URI that you copied into the **Key URI** field.

The screenshot shows the 'Encryption' settings page for a storage account. The 'Use your own key' checkbox is checked. Under 'Encryption key', the 'Enter key URI' radio button is selected, while 'Select from Key Vault' is unselected. The 'Key URI' field contains the value '<key-uri>'. A note at the bottom states: '<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.' with a 'Learn more' link.

5. Specify the subscription that contains the key vault.

6. Save your changes.

## Specify a key from a key vault

To specify a key from a key vault, first make sure that you have a key vault that contains a key. To specify a key

from a key vault, follow these steps:

1. Choose the **Select from Key Vault** option.
2. Select the key vault containing the key you want to use.
3. Select the key from the key vault.

The screenshot shows the 'Encryption key' section of the Azure Storage account settings. At the top, there are 'Save' and 'Discard' buttons. Below them is a note about storage service encryption. Under 'Encryption key', the 'Use your own key' checkbox is checked. There are three options: 'Enter key URI', 'Select from Key Vault' (which is selected), and 'Key Vault'. A note below says 'Your storage account is currently encrypted with Microsoft managed key by default. You can choose to use your own key.' A 'Learn more' link is provided. The 'Select from Key Vault' section shows a dropdown menu with '<key-vault>' and '<key>'. A note at the bottom states: '<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. [Learn more](#)'.

4. Save your changes.

## Update the key version

When you create a new version of a key, update the storage account to use the new version. Follow these steps:

1. Navigate to your storage account and display the **Encryption** settings.
2. Enter the URI for the new key version. Alternately, you can select the key vault and the key again to update the version.
3. Save your changes.

## Use a different key

To change the key used for Azure Storage encryption, follow these steps:

1. Navigate to your storage account and display the **Encryption** settings.
2. Enter the URI for the new key. Alternately, you can select the key vault and choose a new key.
3. Save your changes.

## Disable customer-managed keys

When you disable customer-managed keys, your storage account is once again encrypted with Microsoft-managed keys. To disable customer-managed keys, 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](#)
- [What is Azure Key Vault?](#)



# Configure customer-managed keys with Azure Key Vault by using PowerShell

4/16/2020 • 4 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 supply customer-managed keys to use for encryption of blob and file data.

Customer-managed keys must be stored in an Azure Key Vault. You can either create your own keys and store them in a key vault, or you can use the Azure Key Vault APIs to generate keys. The storage account and the key vault must be in the same region, but they can be in different subscriptions. For more information about Azure Storage encryption and key management, see [Azure Storage encryption for data at rest](#). For more information about Azure Key Vault, see [What is Azure Key Vault?](#)

This article shows how to configure an Azure Key Vault with customer-managed keys using PowerShell. To learn how to create a key vault using Azure CLI, see [Quickstart: Set and retrieve a secret from Azure Key Vault using PowerShell](#).

## Assign an identity to the storage account

To enable customer-managed keys for your 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 key vault.

To assign a managed identity using PowerShell, call [Set-AzStorageAccount](#). Remember to replace the placeholder values in brackets with your own values.

```
$storageAccount = Set-AzStorageAccount -ResourceGroupName <resource_group> `
-Name <storage-account> `
-AssignIdentity
```

For more information about configuring system-assigned managed identities with PowerShell, see [Configure managed identities for Azure resources on an Azure VM using PowerShell](#).

## Create a new key vault

To create a new key vault using PowerShell, call [New-AzKeyVault](#). The key vault that you use to store customer-managed keys for Azure Storage encryption must have two key protection settings enabled, **Soft Delete** and **Do Not Purge**.

Remember to replace the placeholder values in brackets with your own values.

```
$keyVault = New-AzKeyVault -Name <key-vault> `
-ResourceGroupName <resource_group> `
-Location <location> `
-EnableSoftDelete `
-EnablePurgeProtection
```

To learn how to enable **Soft Delete** and **Do Not Purge** on an existing key vault with PowerShell, see the sections titled [Enabling soft-delete](#) and [Enabling Purge Protection](#) in [How to use soft-delete with PowerShell](#).

## Configure the key vault access policy

Next, configure the access policy for the key vault so that the storage account has permissions to access it. In this step, you'll use the managed identity that you previously assigned to the storage account.

To set the access policy for the key vault, call [Set-AzKeyVaultAccessPolicy](#). Remember to replace the placeholder values in brackets with your own values and to use the variables defined in the previous examples.

```
Set-AzKeyVaultAccessPolicy `
 -VaultName $keyVault.VaultName `
 -ObjectId $storageAccount.Identity.PrincipalId `
 -PermissionsToKeys wrapkey,unwrapkey,get
```

## Create a new key

Next, create a new key in the key vault. To create a new key, call [Add-AzKeyVaultKey](#). Remember to replace the placeholder values in brackets with your own values and to use the variables defined in the previous examples.

```
$key = Add-AzKeyVaultKey -VaultName $keyVault.VaultName -Name <key> -Destination 'Software'
```

Only 2048-bit RSA and RSA-HSM keys are supported with Azure Storage encryption. For more information about keys, see [Key Vault keys](#) in [About Azure Key Vault keys, secrets and certificates](#).

## Configure encryption with customer-managed keys

By default, Azure Storage encryption uses Microsoft-managed keys. In this step, configure your Azure Storage account to use customer-managed keys and specify the key to associate with the storage account.

Call [Set-AzStorageAccount](#) to update the storage account's encryption settings, as shown in the following example. Include the **-KeyvaultEncryption** option to enable customer-managed keys for the storage account. Remember to replace the placeholder values in brackets with your own values and to use the variables defined in the previous examples.

```
Set-AzStorageAccount -ResourceGroupName $storageAccount.ResourceGroupName `
 -AccountName $storageAccount.StorageAccountName `
 -KeyvaultEncryption `
 -KeyName $key.Name `
 -KeyVersion $key.Version `
 -KeyVaultUri $keyVault.VaultUri
```

## Update the key version

When you create a new version of a key, you'll need to update the storage account to use the new version. First, call [Get-AzKeyVaultKey](#) to get the latest version of the key. Then call [Set-AzStorageAccount](#) to update the storage account's encryption settings to use the new version of the key, as shown in the previous section.

## Use a different key

To change the key used for Azure Storage encryption, call [Set-AzStorageAccount](#) as shown in [Configure encryption with customer-managed keys](#) and provide the new key name and version. If the new key is in a different key vault, also update the key vault URI.

## Revoke customer-managed keys

If you believe that a key may have been compromised, you can revoke customer-managed keys by removing the key vault access policy. To revoke a customer-managed key, call the [Remove-AzKeyVaultAccessPolicy](#) command, as shown in the following example. Remember to replace the placeholder values in brackets with your own values and to use the variables defined in the previous examples.

```
Remove-AzKeyVaultAccessPolicy -VaultName $keyVault.VaultName `
-ObjectId $storageAccount.Identity.PrincipalId `
```

## Disable customer-managed keys

When you disable customer-managed keys, your storage account is once again encrypted with Microsoft-managed keys. To disable customer-managed keys, call [Set-AzStorageAccount](#) with the `-StorageEncryption` option, as shown in the following example. Remember to replace the placeholder values in brackets with your own values and to use the variables defined in the previous examples.

```
Set-AzStorageAccount -ResourceGroupName $storageAccount.ResourceGroupName `
-AccountName $storageAccount.StorageAccountName `
-StorageEncryption
```

## Next steps

- [Azure Storage encryption for data at rest](#)
- [What is Azure Key Vault?](#)

# Configure customer-managed keys with Azure Key Vault by using Azure CLI

4/16/2020 • 5 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 supply customer-managed keys to use for encryption of blob and file data.

Customer-managed keys must be stored in an Azure Key Vault. You can either create your own keys and store them in a key vault, or you can use the Azure Key Vault APIs to generate keys. The storage account and the key vault must be in the same region, but they can be in different subscriptions. For more information about Azure Storage encryption and key management, see [Azure Storage encryption for data at rest](#). For more information about Azure Key Vault, see [What is Azure Key Vault?](#)

This article shows how to configure an Azure Key Vault with customer-managed keys using Azure CLI. To learn how to create a key vault using Azure CLI, see [Quickstart: Set and retrieve a secret from Azure Key Vault using Azure CLI](#).

## Assign an identity to the storage account

To enable customer-managed keys for your 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 key vault.

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 account set --subscription <subscription-id>

az storage account update \
 --name <storage-account> \
 --resource-group <resource_group> \
 --assign-identity
```

For more information about configuring system-assigned managed identities with Azure CLI, see [Configure managed identities for Azure resources on an Azure VM using Azure CLI](#).

## Create a new key vault

The key vault that you use to store customer-managed keys for Azure Storage encryption must have two key protection settings enabled, **Soft Delete** and **Do Not Purge**. To create a new key vault using PowerShell or Azure CLI with these settings enabled, execute the following commands. Remember to replace the placeholder values in brackets with your own values.

To create a new key vault using Azure CLI, call `az keyvault create`. Remember to replace the placeholder values in brackets with your own values.

```
az keyvault create \
--name <key-vault> \
--resource-group <resource_group> \
--location <region> \
--enable-soft-delete \
--enable-purge-protection
```

To learn how to enable **Soft Delete** and **Do Not Purge** on an existing key vault with Azure CLI, see the sections titled **Enabling soft-delete** and **Enabling Purge Protection** in [How to use soft-delete with CLI](#).

## Configure the key vault access policy

Next, configure the access policy for the key vault so that the storage account has permissions to access it. In this step, you'll use the managed identity that you previously assigned to the storage account.

To set the access policy for the key vault, call [az keyvault set-policy](#). 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 set-policy \
--name <key-vault> \
--resource-group <resource_group>
--object-id $storage_account_principal \
--key-permissions get unwrapKey wrapKey
```

## Create a new key

Next, create a key in the key vault. To create a key, call [az keyvault key create](#). Remember to replace the placeholder values in brackets with your own values.

```
az keyvault key create
--name <key> \
--vault-name <key-vault>
```

Only 2048-bit RSA and RSA-HSM keys are supported with Azure Storage encryption. For more information about keys, see **Key Vault keys** in [About Azure Key Vault keys, secrets and certificates](#).

## Configure encryption with customer-managed keys

By default, Azure Storage encryption uses Microsoft-managed keys. Configure your Azure Storage account for customer-managed keys and specify the key to associate with the storage account.

To update the storage account's encryption settings, call [az storage account update](#), 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 storage account. The example also queries for the key vault URI and the latest key version, both of which values are needed to associate the key with the storage account. Remember to replace the placeholder values in brackets with your own values.

```
key_vault_uri=$(az keyvault show \
 --name <key-vault> \
 --resource-group <resource_group> \
 --query properties.vaultUri \
 --output tsv)
key_version=$(az keyvault key list-versions \
 --name <key> \
 --vault-name <key-vault> \
 --query [-1].kid \
 --output tsv | cut -d '/' -f 6)
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 $key_vault_uri
```

## Update the key version

When you create a new version of a key, you'll need to update the storage account to use the new version. First, query for the key vault URL 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 section.

## Use a different key

To change the key used for Azure Storage encryption, call [az storage account update](#) as shown in [Configure encryption with customer-managed keys](#) and provide the new key name and version. If the new key is in a different key vault, also update the key vault URL.

## Revoke customer-managed keys

If you believe that a key may have been compromised, you can revoke customer-managed keys by removing the key vault access policy. To revoke a customer-managed key, call the [az keyvault delete-policy](#) command, as shown in the following example. Remember to replace the placeholder values in brackets with your own values and to use the variables defined in the previous examples.

```
az keyvault delete-policy \
 --name <key-vault> \
 --object-id $storage_account_principal
```

## Disable customer-managed keys

When you disable customer-managed keys, your storage account is once again encrypted with Microsoft-managed keys. To disable customer-managed keys, call [az storage account update](#) and set the `--encryption-key-source` parameter to `Microsoft.Storage`, as shown in the following example. Remember to replace the placeholder values in brackets with your own values and to use the variables defined in the previous examples.

```
az storage account update \
 --name <storage-account> \
 --resource-group <resource_group> \
 --encryption-key-source Microsoft.Storage
```

## Next steps

- [Azure Storage encryption for data at rest](#)
- [What is Azure Key Vault?](#)

# Client-Side Encryption and Azure Key Vault for Microsoft Azure Storage

4/16/2020 • 13 minutes to read • [Edit Online](#)

## Overview

The [Azure 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.

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](#).

For client-side encryption with Java, see [Client-Side Encryption with Java for Microsoft Azure Storage](#).

## Encryption and decryption via the envelope technique

The processes of encryption and decryption follow the envelope technique.

### Encryption via the envelope technique

Encryption via the envelope technique works in the following way:

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 this 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 an asymmetric key pair or a symmetric key and can be managed locally or stored in Azure Key Vaults.

The 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 the Azure Storage service. The wrapped key along with some additional encryption metadata is either stored as metadata (on a blob) or interpolated with the encrypted data (queue messages and table entities).

### Decryption via the envelope technique

Decryption via the envelope technique works in the following way:

1. The client library assumes that the user is managing the key encryption key (KEK) either locally or in Azure Key Vaults. The user does not need to know the specific key that was used for encryption. Instead, a key resolver which 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 on the service.
3. The wrapped content encryption key (CEK) is then unwrapped (decrypted) using the key encryption key (KEK). Here again, the client library does not have access to KEK. It simply invokes the custom or Key Vault provider's unwrapping algorithm.
4. The content encryption key (CEK) is then used to decrypt the encrypted user data.

# Encryption Mechanism

The storage client library uses [AES](#) in order to encrypt user data. Specifically, [Cipher Block Chaining \(CBC\)](#) mode with AES. Each service works somewhat differently, so we will discuss each of them here.

## Blobs

The client library currently supports encryption of whole blobs only. Specifically, encryption is supported when users use the [UploadFrom](#) methods or the [OpenWrite](#) method. For downloads, both complete and range downloads are supported.

During encryption, the client library will generate a random Initialization Vector (IV) of 16 bytes, together with a random content encryption key (CEK) of 32 bytes, and perform 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 on the service.

### WARNING

If you are editing or uploading your own metadata for the blob, you need to ensure that this metadata is preserved. If you upload new metadata without this metadata, the wrapped CEK, IV, and other metadata will be lost and the blob content will never be retrievable again.

Downloading an encrypted blob involves retrieving the content of the entire blob using the [DownloadTo/BlobReadStream](#) convenience methods. The wrapped CEK is unwrapped and used together with the IV (stored as blob metadata in this case) to return the decrypted data to the users.

Downloading an arbitrary range ([DownloadRange](#) methods) 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.

## Queues

Since queue messages can be of any format, the client library defines a custom format that includes the Initialization Vector (IV) and the encrypted content encryption key (CEK) in the message text.

During encryption, the client library generates a random IV of 16 bytes along with a random CEK of 32 bytes and performs envelope encryption of the queue message text using this information. The wrapped CEK and some additional encryption metadata are then added to the encrypted queue message. This modified message (shown below) is stored on the service.

```
<MessageText>
{"EncryptedMessageContents": "6k0u8Rq1C3+M1Q04a1KLmWthWXSmHV3mEfxbAgP9QGTU++MKn2uPq3t2UjF1D06w", "EncryptionData": {...}}</MessageText>
```

During decryption, the wrapped key is extracted from the queue message and unwrapped. The IV is also extracted from the queue message and used along with the unwrapped key to decrypt the queue message data. Note that the encryption metadata is small (under 500 bytes), so while it does count toward the 64KB limit for a queue message, the impact should be manageable.

## Tables

The client library supports encryption of entity properties for insert and replace operations.

#### **NOTE**

Merge is not currently supported. Since a subset of properties may have been encrypted previously using a different key, simply merging the new properties and updating the metadata will result in data loss. Merging either requires making extra service calls to read the pre-existing entity from the service, or using a new key per property, both of which are not suitable for performance reasons.

Table data encryption works as follows:

1. Users specify the properties to be encrypted.
2. The client library generates a random Initialization Vector (IV) of 16 bytes along with a random content encryption key (CEK) of 32 bytes for every entity, and performs envelope encryption on the individual properties to be encrypted by deriving a new IV per property. The encrypted property is stored as binary data.
3. The wrapped CEK and some additional encryption metadata are then stored as two additional reserved properties. The first reserved property (`_ClientEncryptionMetadata1`) is a string property that holds the information about IV, version, and wrapped key. The second reserved property (`_ClientEncryptionMetadata2`) is a binary property that holds the information about the properties that are encrypted. The information in this second property (`_ClientEncryptionMetadata2`) is itself encrypted.
4. Due to these additional reserved properties required for encryption, users may now have only 250 custom properties instead of 252. The total size of the entity must be less than 1 MB.

Note that only string properties can be encrypted. If other types of properties are to be encrypted, they must be converted to strings. The encrypted strings are stored on the service as binary properties, and they are converted back to strings after decryption.

For tables, in addition to the encryption policy, users must specify the properties to be encrypted. This can be done by either specifying an `[EncryptProperty]` attribute (for POCO entities that derive from `TableEntity`) or an encryption resolver in request options. An encryption resolver is a delegate that takes a partition key, row key, and property name and returns a Boolean that indicates whether that property should be encrypted. During encryption, the client library will use this information to decide whether a property should be encrypted while writing to the wire. The delegate also provides for the possibility of logic around how properties are encrypted. (For example, if X, then encrypt property A; otherwise encrypt properties A and B.) Note that it is not necessary to provide this information while reading or querying entities.

### **Batch Operations**

In batch operations, the same KEK will be used across all the rows in that batch operation because the client library only allows one options object (and hence one policy/KEK) per batch operation. However, the client library will internally generate a new random IV and random CEK per row in the batch. Users can also choose to encrypt different properties for every operation in the batch by defining this behavior in the encryption resolver.

### **Queries**

#### **NOTE**

Because the entities are encrypted, you cannot run queries that filter on an encrypted property. If you try, results will be incorrect, because the service would be trying to compare encrypted data with unencrypted data.

To perform query operations, you must specify a key resolver that is able to resolve all the keys in the result set. If an entity contained in the query result cannot be resolved to a provider, the client library will throw an error. For any query that performs server-side projections, the client library will add the special encryption metadata properties (`_ClientEncryptionMetadata1` and `_ClientEncryptionMetadata2`) by default to the selected columns.

# Azure Key Vault

Azure Key Vault helps safeguard cryptographic keys and secrets used by cloud applications and services. By using Azure Key Vault, users can encrypt keys and secrets (such as authentication keys, storage account keys, data encryption keys, .PFX files, and passwords) by using keys that are protected by hardware security modules (HSMs). For more information, see [What is Azure Key Vault?](#).

The storage client library uses the Key Vault core library in order to provide a common framework across Azure for managing keys. Users also get the additional benefit of using the Key Vault extensions library. The extensions library provides useful functionality around simple and seamless Symmetric/RSA local and cloud key providers as well as with aggregation and caching.

## Interface and dependencies

There are three Key Vault packages:

- Microsoft.Azure.KeyVault.Core contains the IKey and IKeyResolver. It is a small package with no dependencies. The storage client library for .NET defines it as a dependency.
- Microsoft.Azure.KeyVault contains the Key Vault REST client.
- Microsoft.Azure.KeyVault.Extensions contains extension code that includes implementations of cryptographic algorithms and an RSAKey and a SymmetricKey. It depends on the Core and KeyVault namespaces and provides functionality to define an aggregate resolver (when users want to use multiple key providers) and a caching key resolver. Although the storage client library does not directly depend on this package, if users wish to use Azure Key Vault to store their keys or to use the Key Vault extensions to consume the local and cloud cryptographic providers, they will need this package.

Key Vault is designed for high-value master keys, and throttling limits per Key Vault are designed with this in mind. When performing client-side encryption with Key Vault, the preferred model is to use symmetric master keys stored as secrets in Key Vault and cached locally. Users must do the following:

1. Create a secret offline and upload it to Key Vault.
2. Use the secret's base identifier as a parameter to resolve the current version of the secret for encryption and cache this information locally. Use CachingKeyResolver for caching; users are not expected to implement their own caching logic.
3. Use the caching resolver as an input while creating the encryption policy.

More information regarding Key Vault usage can be found in the [encryption code samples](#).

## Best practices

Encryption support is available only in the storage client library for .NET. Windows Phone and Windows Runtime do not currently support encryption.

## IMPORTANT

Be aware of these important points when using client-side encryption:

- When reading from or writing to an encrypted blob, use whole blob upload commands and range/whole blob download commands. Avoid writing to an encrypted blob using protocol operations such as Put Block, Put Block List, Write Pages, Clear Pages, or Append Block; otherwise you may corrupt the encrypted blob and make it unreadable.
- For tables, a similar constraint exists. Be careful to not update encrypted properties without updating the encryption metadata.
- If you set metadata on the encrypted blob, you may overwrite the encryption-related metadata required for decryption, since setting metadata is not additive. This is also true for snapshots; avoid specifying metadata while creating a snapshot of an encrypted blob. If metadata must be set, be sure to call the **FetchAttributes** method first to get the current encryption metadata, and avoid concurrent writes while metadata is being set.
- Enable the **RequireEncryption** property in the default request options for users that should work only with encrypted data. See below for more info.

## Client API / Interface

While creating an **EncryptionPolicy** object, users can provide only a Key (implementing **IKey**), only a resolver (implementing **IKeyResolver**), or both. **IKey** is the basic key type that is identified using a key identifier and that provides the logic for wrapping/unwrapping. **IKeyResolver** is used to resolve a key during the decryption process. It defines a **ResolveKey** method that returns an **IKey** given a key identifier. This provides users the ability to choose between multiple keys that are managed in multiple locations.

- For encryption, the key is used always and the absence of a key will result in an error.
- For decryption:
  - The key resolver is invoked if specified to get the key. If the resolver is specified but does not have a mapping for the key identifier, an error is thrown.
  - If resolver is not specified but a key is specified, the key is used if its identifier matches the required key identifier. If the identifier does not match, an error is thrown.

The code examples in this article demonstrate setting an encryption policy and working with encrypted data, but do not demonstrate working with Azure Key Vault. The [encryption samples](#) on GitHub demonstrate a more detailed end-to-end scenario for blobs, queues and tables, along with Key Vault integration.

### RequireEncryption mode

Users can optionally enable a mode of operation where all uploads and downloads must be encrypted. In this mode, attempts to upload data without an encryption policy or download data that is not encrypted on the service will fail on the client. The **RequireEncryption** property of the request options object controls this behavior. If your application will encrypt all objects stored in Azure Storage, then you can set the **RequireEncryption** property on the default request options for the service client object. For example, set **CloudBlobClient.DefaultRequestOptions.RequireEncryption** to **true** to require encryption for all blob operations performed through that client object.

### Blob service encryption

Create a **BlobEncryptionPolicy** object and set it in the request options (per API or at a client level by using **DefaultRequestOptions**). Everything else will be handled by the client library internally.

```

// Create the IKey used for encryption.
RsaKey key = new RsaKey("private:key1" /* key identifier */);

// Create the encryption policy to be used for upload and download.
BlobEncryptionPolicy policy = new BlobEncryptionPolicy(key, null);

// Set the encryption policy on the request options.
BlobRequestOptions options = new BlobRequestOptions() { EncryptionPolicy = policy };

// Upload the encrypted contents to the blob.
blob.UploadFromStream(stream, size, null, options, null);

// Download and decrypt the encrypted contents from the blob.
MemoryStream outputStream = new MemoryStream();
blob.DownloadToStream(outputStream, null, options, null);

```

## Queue service encryption

Create a `QueueEncryptionPolicy` object and set it in the request options (per API or at a client level by using `DefaultRequestOptions`). Everything else will be handled by the client library internally.

```

// Create the IKey used for encryption.
RsaKey key = new RsaKey("private:key1" /* key identifier */);

// Create the encryption policy to be used for upload and download.
QueueEncryptionPolicy policy = new QueueEncryptionPolicy(key, null);

// Add message
QueueRequestOptions options = new QueueRequestOptions() { EncryptionPolicy = policy };
queue.AddMessage(message, null, null, options, null);

// Retrieve message
CloudQueueMessage retrMessage = queue.GetMessage(null, options, null);

```

## Table service encryption

In addition to creating an encryption policy and setting it on request options, you must either specify an `EncryptionResolver` in `TableRequestOptions`, or set the `[EncryptProperty]` attribute on the entity.

### Using the resolver

```

// Create the IKey used for encryption.
RsaKey key = new RsaKey("private:key1" /* key identifier */);

// Create the encryption policy to be used for upload and download.
TableEncryptionPolicy policy = new TableEncryptionPolicy(key, null);

TableRequestOptions options = new TableRequestOptions()
{
 EncryptionResolver = (pk, rk, propName) =>
 {
 if (propName == "foo")
 {
 return true;
 }
 return false;
 },
 EncryptionPolicy = policy
};

// Insert Entity
currentTable.Execute(TableOperation.Insert(ent), options, null);

// Retrieve Entity
// No need to specify an encryption resolver for retrieve
TableRequestOptions retrieveOptions = new TableRequestOptions()
{
 EncryptionPolicy = policy
};

TableOperation operation = TableOperation.Retrieve(ent.PartitionKey, ent.RowKey);
TableResult result = currentTable.Execute(operation, retrieveOptions, null);

```

#### Using attributes

As mentioned above, if the entity implements `TableEntity`, then the properties can be decorated with the `[EncryptProperty]` attribute instead of specifying the `EncryptionResolver`.

```

[EncryptProperty]
public string EncryptedProperty1 { get; set; }

```

## Encryption and performance

Note that encrypting your storage data results in additional performance overhead. The content key and IV must be generated, the content itself must be encrypted, and additional meta-data must be formatted and uploaded. This overhead will vary depending on the quantity of data being encrypted. We recommend that customers always test their applications for performance during development.

## Next steps

- [Tutorial: Encrypt and decrypt blobs in Microsoft Azure Storage using Azure Key Vault](#)
- Download the [Azure Storage Client Library for .NET NuGet package](#)
- Download the Azure Key Vault NuGet [Core](#), [Client](#), and [Extensions](#) packages
- Visit the [Azure Key Vault Documentation](#)

# Client-Side Encryption and Azure Key Vault with Java for Microsoft Azure Storage

4/16/2020 • 13 minutes to read • [Edit Online](#)

## Overview

The [Azure Storage Client Library for Java](#) 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.

## Encryption and decryption via the envelope technique

The processes of encryption and decryption follow the envelope technique.

### Encryption via the envelope technique

Encryption via the envelope technique works in the following way:

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 this 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 an asymmetric key pair or a symmetric key and can be managed locally or stored in Azure Key Vaults.  
The 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 the Azure Storage service. The wrapped key along with some additional encryption metadata is either stored as metadata (on a blob) or interpolated with the encrypted data (queue messages and table entities).

### Decryption via the envelope technique

Decryption via the envelope technique works in the following way:

1. The client library assumes that the user is managing the key encryption key (KEK) either locally or in Azure Key Vaults. The user does not need to know the specific key that was used for encryption. Instead, a key resolver which 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 on the service.
3. The wrapped content encryption key (CEK) is then unwrapped (decrypted) using the key encryption key (KEK). Here again, the client library does not have access to KEK. It simply invokes the custom or Key Vault provider's unwrapping algorithm.
4. The content encryption key (CEK) is then used to decrypt the encrypted user data.

## Encryption Mechanism

The storage client library uses [AES](#) in order to encrypt user data. Specifically, [Cipher Block Chaining \(CBC\)](#) mode with AES. Each service works somewhat differently, so we will discuss each of them here.

### Blobs

The client library currently supports encryption of whole blobs only. Specifically, encryption is supported when

users use the `upload*` methods or the `openOutputStream` method. For downloads, both complete and range downloads are supported.

During encryption, the client library will generate a random Initialization Vector (IV) of 16 bytes, together with a random content encryption key (CEK) of 32 bytes, and perform 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 on the service.

#### WARNING

If you are editing or uploading your own metadata for the blob, you need to ensure that this metadata is preserved. If you upload new metadata without this metadata, the wrapped CEK, IV and other metadata will be lost and the blob content will never be retrievable again.

Downloading an encrypted blob involves retrieving the content of the entire blob using the `download/openInputStream` convenience methods. The wrapped CEK is unwrapped and used together with the IV (stored as blob metadata in this case) to return the decrypted data to the users.

Downloading an arbitrary range (`downloadRange` methods) 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.

#### Queues

Since queue messages can be of any format, the client library defines a custom format that includes the Initialization Vector (IV) and the encrypted content encryption key (CEK) in the message text.

During encryption, the client library generates a random IV of 16 bytes along with a random CEK of 32 bytes and performs envelope encryption of the queue message text using this information. The wrapped CEK and some additional encryption metadata are then added to the encrypted queue message. This modified message (shown below) is stored on the service.

```
<MessageText>
{"EncryptedMessageContents": "6k0u8Rq1C3+M1Q04a1KLmWthWXSmHV3mEfxBAgP9QGTU++MKn2uPq3t2UjF1D06w", "EncryptionData": {...}}</MessageText>
```

During decryption, the wrapped key is extracted from the queue message and unwrapped. The IV is also extracted from the queue message and used along with the unwrapped key to decrypt the queue message data. Note that the encryption metadata is small (under 500 bytes), so while it does count toward the 64KB limit for a queue message, the impact should be manageable.

#### Tables

The client library supports encryption of entity properties for insert and replace operations.

#### NOTE

Merge is not currently supported. Since a subset of properties may have been encrypted previously using a different key, simply merging the new properties and updating the metadata will result in data loss. Merging either requires making extra service calls to read the pre-existing entity from the service, or using a new key per property, both of which are not suitable for performance reasons.

Table data encryption works as follows:

1. Users specify the properties to be encrypted.

2. The client library generates a random Initialization Vector (IV) of 16 bytes along with a random content encryption key (CEK) of 32 bytes for every entity, and performs envelope encryption on the individual properties to be encrypted by deriving a new IV per property. The encrypted property is stored as binary data.
3. The wrapped CEK and some additional encryption metadata are then stored as two additional reserved properties. The first reserved property (`_ClientEncryptionMetadata1`) is a string property that holds the information about IV, version, and wrapped key. The second reserved property (`_ClientEncryptionMetadata2`) is a binary property that holds the information about the properties that are encrypted. The information in this second property (`_ClientEncryptionMetadata2`) is itself encrypted.
4. Due to these additional reserved properties required for encryption, users may now have only 250 custom properties instead of 252. The total size of the entity must be less than 1MB.

Note that only string properties can be encrypted. If other types of properties are to be encrypted, they must be converted to strings. The encrypted strings are stored on the service as binary properties, and they are converted back to strings after decryption.

For tables, in addition to the encryption policy, users must specify the properties to be encrypted. This can be done by either specifying an [Encrypt] attribute (for POCO entities that derive from `TableEntity`) or an encryption resolver in request options. An encryption resolver is a delegate that takes a partition key, row key, and property name and returns a boolean that indicates whether that property should be encrypted. During encryption, the client library will use this information to decide whether a property should be encrypted while writing to the wire. The delegate also provides for the possibility of logic around how properties are encrypted. (For example, if X, then encrypt property A; otherwise encrypt properties A and B.) Note that it is not necessary to provide this information while reading or querying entities.

## Batch Operations

In batch operations, the same KEK will be used across all the rows in that batch operation because the client library only allows one options object (and hence one policy/KEK) per batch operation. However, the client library will internally generate a new random IV and random CEK per row in the batch. Users can also choose to encrypt different properties for every operation in the batch by defining this behavior in the encryption resolver.

## Queries

### NOTE

Because the entities are encrypted, you cannot run queries that filter on an encrypted property. If you try, results will be incorrect, because the service would be trying to compare encrypted data with unencrypted data.

To perform query operations, you must specify a key resolver that is able to resolve all the keys in the result set. If an entity contained in the query result cannot be resolved to a provider, the client library will throw an error. For any query that performs server side projections, the client library will add the special encryption metadata properties (`_ClientEncryptionMetadata1` and `_ClientEncryptionMetadata2`) by default to the selected columns.

## Azure Key Vault

Azure Key Vault helps safeguard cryptographic keys and secrets used by cloud applications and services. By using Azure Key Vault, users can encrypt keys and secrets (such as authentication keys, storage account keys, data encryption keys, .PFX files, and passwords) by using keys that are protected by hardware security modules (HSMs). For more information, see [What is Azure Key Vault?](#).

The storage client library uses the Key Vault core library in order to provide a common framework across Azure for managing keys. Users also get the additional benefit of using the Key Vault extensions library. The extensions library provides useful functionality around simple and seamless Symmetric/RSA local and cloud key providers as well as with aggregation and caching.

## Interface and dependencies

There are three Key Vault packages:

- `azure-keyvault-core` contains the `IKey` and `IKeyResolver`. It is a small package with no dependencies. The storage client library for Java defines it as a dependency.
- `azure-keyvault` contains the Key Vault REST client.
- `azure-keyvault-extensions` contains extension code that includes implementations of cryptographic algorithms and an `RSAKey` and a `SymmetricKey`. It depends on the Core and KeyVault namespaces and provides functionality to define an aggregate resolver (when users want to use multiple key providers) and a caching key resolver. Although the storage client library does not directly depend on this package, if users wish to use Azure Key Vault to store their keys or to use the Key Vault extensions to consume the local and cloud cryptographic providers, they will need this package.

Key Vault is designed for high-value master keys, and throttling limits per Key Vault are designed with this in mind. When performing client-side encryption with Key Vault, the preferred model is to use symmetric master keys stored as secrets in Key Vault and cached locally. Users must do the following:

1. Create a secret offline and upload it to Key Vault.
2. Use the secret's base identifier as a parameter to resolve the current version of the secret for encryption and cache this information locally. Use `CachingKeyResolver` for caching; users are not expected to implement their own caching logic.
3. Use the caching resolver as an input while creating the encryption policy. More information regarding Key Vault usage can be found in the encryption code samples.

## Best practices

Encryption support is available only in the storage client library for Java.

### IMPORTANT

Be aware of these important points when using client-side encryption:

- When reading from or writing to an encrypted blob, use whole blob upload commands and range/whole blob download commands. Avoid writing to an encrypted blob using protocol operations such as Put Block, Put Block List, Write Pages, Clear Pages, or Append Block; otherwise you may corrupt the encrypted blob and make it unreadable.
- For tables, a similar constraint exists. Be careful to not update encrypted properties without updating the encryption metadata.
- If you set metadata on the encrypted blob, you may overwrite the encryption-related metadata required for decryption, since setting metadata is not additive. This is also true for snapshots; avoid specifying metadata while creating a snapshot of an encrypted blob. If metadata must be set, be sure to call the `downloadAttributes` method first to get the current encryption metadata, and avoid concurrent writes while metadata is being set.
- Enable the `requireEncryption` flag in the default request options for users that should work only with encrypted data. See below for more info.

## Client API / Interface

While creating an `EncryptionPolicy` object, users can provide only a Key (implementing `IKey`), only a resolver (implementing `IKeyResolver`), or both. `IKey` is the basic key type that is identified using a key identifier and that provides the logic for wrapping/unwrapping. `IKeyResolver` is used to resolve a key during the decryption process. It defines a `ResolveKey` method that returns an `IKey` given a key identifier. This provides users the ability to choose between multiple keys that are managed in multiple locations.

- For encryption, the key is used always and the absence of a key will result in an error.

- For decryption:
  - The key resolver is invoked if specified to get the key. If the resolver is specified but does not have a mapping for the key identifier, an error is thrown.
  - If resolver is not specified but a key is specified, the key is used if its identifier matches the required key identifier. If the identifier does not match, an error is thrown.

The [encryption samples](#) demonstrate a more detailed end-to-end scenario for blobs, queues and tables, along with Key Vault integration.

## RequireEncryption mode

Users can optionally enable a mode of operation where all uploads and downloads must be encrypted. In this mode, attempts to upload data without an encryption policy or download data that is not encrypted on the service will fail on the client. The **requireEncryption** flag of the request options object controls this behavior. If your application will encrypt all objects stored in Azure Storage, then you can set the **requireEncryption** property on the default request options for the service client object.

For example, use `CloudBlobClient.getDefaultRequestOptions().setRequireEncryption(true)` to require encryption for all blob operations performed through that client object.

## Blob service encryption

Create a **BlobEncryptionPolicy** object and set it in the request options (per API or at a client level by using **DefaultRequestOptions**). Everything else will be handled by the client library internally.

```
// Create the IKey used for encryption.
RsaKey key = new RsaKey("private:key1" /* key identifier */);

// Create the encryption policy to be used for upload and download.
BlobEncryptionPolicy policy = new BlobEncryptionPolicy(key, null);

// Set the encryption policy on the request options.
BlobRequestOptions options = new BlobRequestOptions();
options.setEncryptionPolicy(policy);

// Upload the encrypted contents to the blob.
blob.upload(stream, size, null, options, null);

// Download and decrypt the encrypted contents from the blob.
ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
blob.download(outputStream, null, options, null);
```

## Queue service encryption

Create a **QueueEncryptionPolicy** object and set it in the request options (per API or at a client level by using **DefaultRequestOptions**). Everything else will be handled by the client library internally.

```

// Create the IKey used for encryption.
RsaKey key = new RsaKey("private:key1" /* key identifier */);

// Create the encryption policy to be used for upload and download.
QueueEncryptionPolicy policy = new QueueEncryptionPolicy(key, null);

// Add message
QueueRequestOptions options = new QueueRequestOptions();
options.setEncryptionPolicy(policy);

queue.addMessage(message, 0, 0, options, null);

// Retrieve message
CloudQueueMessage retrMessage = queue.retrieveMessage(30, options, null);

```

## Table service encryption

In addition to creating an encryption policy and setting it on request options, you must either specify an **EncryptionResolver** in **TableRequestOptions**, or set the **[Encrypt]** attribute on the entity's getter and setter.

## Using the resolver

```

// Create the IKey used for encryption.
RsaKey key = new RsaKey("private:key1" /* key identifier */);

// Create the encryption policy to be used for upload and download.
TableEncryptionPolicy policy = new TableEncryptionPolicy(key, null);

TableRequestOptions options = new TableRequestOptions()
options.setEncryptionPolicy(policy);
options.setEncryptionResolver(new EncryptionResolver() {
 public boolean encryptionResolver(String pk, String rk, String key) {
 if (key == "foo")
 {
 return true;
 }
 return false;
 }
});

// Insert Entity
currentTable.execute(TableOperation.insert(ent), options, null);

// Retrieve Entity
// No need to specify an encryption resolver for retrieve
TableRequestOptions retrieveOptions = new TableRequestOptions()
retrieveOptions.setEncryptionPolicy(policy);

TableOperation operation = TableOperation.retrieve(ent.PartitionKey, ent.RowKey, DynamicTableEntity.class);
TableResult result = currentTable.execute(operation, retrieveOptions, null);

```

## Using attributes

As mentioned above, if the entity implements **TableEntity**, then the properties getter and setter can be decorated with the **[Encrypt]** attribute instead of specifying the **EncryptionResolver**.

```
private string encryptedProperty1;

@Encrypt
public String getEncryptedProperty1 () {
 return this.encryptedProperty1;
}

@Encrypt
public void setEncryptedProperty1(final String encryptedProperty1) {
 this.encryptedProperty1 = encryptedProperty1;
}
```

## Encryption and performance

Note that encrypting your storage data results in additional performance overhead. The content key and IV must be generated, the content itself must be encrypted, and additional meta-data must be formatted and uploaded. This overhead will vary depending on the quantity of data being encrypted. We recommend that customers always test their applications for performance during development.

## Next steps

- Download the [Azure Storage Client Library for Java Maven package](#)
- Download the [Azure Storage Client Library for Java Source Code from GitHub](#)
- Download the Azure Key Vault Maven Library for Java Maven packages:
  - [Core](#) package
  - [Client](#) package
- Visit the [Azure Key Vault Documentation](#)

# Client-side encryption with Python

12/5/2019 • 13 minutes to read • [Edit Online](#)

## Overview

The [Azure Storage Client Library for Python](#) supports encrypting data within client applications before uploading to Azure Storage, and decrypting data while downloading to the client.

### NOTE

The Azure Storage Python library is in preview.

## Encryption and decryption via the envelope technique

The processes of encryption and decryption follow the envelope technique.

### Encryption via the envelope technique

Encryption via the envelope technique works in the following way:

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 this 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 an asymmetric key pair or a symmetric key, which is managed locally. The storage client library itself never has access to KEK. The library invokes the key wrapping algorithm that is provided by the KEK. Users can choose to use custom providers for key wrapping/unwrapping if desired.
4. The encrypted data is then uploaded to the Azure Storage service. The wrapped key along with some additional encryption metadata is either stored as metadata (on a blob) or interpolated with the encrypted data (queue messages and table entities).

### Decryption via the envelope technique

Decryption via the envelope technique works in the following way:

1. The client library assumes that the user is managing the key encryption key (KEK) locally. The user does not need to know the specific key that was used for encryption. Instead, a key resolver, which 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 on the service.
3. The wrapped content encryption key (CEK) is then unwrapped (decrypted) using the key encryption key (KEK). Here again, the client library does not have access to KEK. It simply invokes the custom provider's unwrapping algorithm.
4. The content encryption key (CEK) is then used to decrypt the encrypted user data.

## Encryption Mechanism

The storage client library uses [AES](#) in order to encrypt user data. Specifically, [Cipher Block Chaining \(CBC\)](#) mode with AES. Each service works somewhat differently, so we will discuss each of them here.

### Blobs

The client library currently supports encryption of whole blobs only. Specifically, encryption is supported when users use the `create*` methods. For downloads, both complete and range downloads are supported, and parallelization of both upload and download is available.

During encryption, the client library will generate a random Initialization Vector (IV) of 16 bytes, together with a random content encryption key (CEK) of 32 bytes, and perform 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 on the service.

#### WARNING

If you are editing or uploading your own metadata for the blob, you need to ensure that this metadata is preserved. If you upload new metadata without this metadata, the wrapped CEK, IV and other metadata will be lost and the blob content will never be retrievable again.

Downloading an encrypted blob involves retrieving the content of the entire blob using the `get*` convenience methods. The wrapped CEK is unwrapped and used together with the IV (stored as blob metadata in this case) to return the decrypted data to the users.

Downloading an arbitrary range (`get*` methods with range parameters passed in) 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.

Block blobs and page blobs only can be encrypted/decrypted using this scheme. There is currently no support for encrypting append blobs.

#### Queues

Since queue messages can be of any format, the client library defines a custom format that includes the Initialization Vector (IV) and the encrypted content encryption key (CEK) in the message text.

During encryption, the client library generates a random IV of 16 bytes along with a random CEK of 32 bytes and performs envelope encryption of the queue message text using this information. The wrapped CEK and some additional encryption metadata are then added to the encrypted queue message. This modified message (shown below) is stored on the service.

```
<MessageText>
{"EncryptedMessageContents": "6k0u8Rq1C3+M1Q04a1KLmWthWXSmHV3mEfxBAgP9QGTU++MKn2uPq3t2UjF1D06w", "EncryptionData": {...}}</MessageText>
```

During decryption, the wrapped key is extracted from the queue message and unwrapped. The IV is also extracted from the queue message and used along with the unwrapped key to decrypt the queue message data. Note that the encryption metadata is small (under 500 bytes), so while it does count toward the 64KB limit for a queue message, the impact should be manageable.

#### Tables

The client library supports encryption of entity properties for insert and replace operations.

#### NOTE

Merge is not currently supported. Since a subset of properties may have been encrypted previously using a different key, simply merging the new properties and updating the metadata will result in data loss. Merging either requires making extra service calls to read the pre-existing entity from the service, or using a new key per property, both of which are not suitable for performance reasons.

Table data encryption works as follows:

1. Users specify the properties to be encrypted.
2. The client library generates a random Initialization Vector (IV) of 16 bytes along with a random content encryption key (CEK) of 32 bytes for every entity, and performs envelope encryption on the individual properties to be encrypted by deriving a new IV per property. The encrypted property is stored as binary data.
3. The wrapped CEK and some additional encryption metadata are then stored as two additional reserved properties. The first reserved property (`_ClientEncryptionMetadata1`) is a string property that holds the information about IV, version, and wrapped key. The second reserved property (`_ClientEncryptionMetadata2`) is a binary property that holds the information about the properties that are encrypted. The information in this second property (`_ClientEncryptionMetadata2`) is itself encrypted.
4. Due to these additional reserved properties required for encryption, users may now have only 250 custom properties instead of 252. The total size of the entity must be less than 1MB.

Note that only string properties can be encrypted. If other types of properties are to be encrypted, they must be converted to strings. The encrypted strings are stored on the service as binary properties, and they are converted back to strings (raw strings, not EntityProperties with type `EdmType.STRING`) after decryption.

For tables, in addition to the encryption policy, users must specify the properties to be encrypted. This can be done by either storing these properties in `TableEntity` objects with the type set to `EdmType.STRING` and encrypt set to true or setting the `encryption_resolver_function` on the `tableservice` object. An encryption resolver is a function that takes a partition key, row key, and property name and returns a boolean that indicates whether that property should be encrypted. During encryption, the client library will use this information to decide whether a property should be encrypted while writing to the wire. The delegate also provides for the possibility of logic around how properties are encrypted. (For example, if X, then encrypt property A; otherwise encrypt properties A and B.) Note that it is not necessary to provide this information while reading or querying entities.

## Batch Operations

One encryption policy applies to all rows in the batch. The client library will internally generate a new random IV and random CEK per row in the batch. Users can also choose to encrypt different properties for every operation in the batch by defining this behavior in the encryption resolver. If a batch is created as a context manager through the `tableservice batch()` method, the `tableservice`'s encryption policy will automatically be applied to the batch. If a batch is created explicitly by calling the constructor, the encryption policy must be passed as a parameter and left unmodified for the lifetime of the batch. Note that entities are encrypted as they are inserted into the batch using the batch's encryption policy (entities are NOT encrypted at the time of committing the batch using the `tableservice`'s encryption policy).

## Queries

### NOTE

Because the entities are encrypted, you cannot run queries that filter on an encrypted property. If you try, results will be incorrect, because the service would be trying to compare encrypted data with unencrypted data.

To perform query operations, you must specify a key resolver that is able to resolve all the keys in the result set. If an entity contained in the query result cannot be resolved to a provider, the client library will throw an error. For any query that performs server side projections, the client library will add the special encryption metadata properties (`_ClientEncryptionMetadata1` and `_ClientEncryptionMetadata2`) by default to the selected columns.

## IMPORTANT

Be aware of these important points when using client-side encryption:

- When reading from or writing to an encrypted blob, use whole blob upload commands and range/whole blob download commands. Avoid writing to an encrypted blob using protocol operations such as Put Block, Put Block List, Write Pages, or Clear Pages; otherwise you may corrupt the encrypted blob and make it unreadable.
- For tables, a similar constraint exists. Be careful to not update encrypted properties without updating the encryption metadata.
- If you set metadata on the encrypted blob, you may overwrite the encryption-related metadata required for decryption, since setting metadata is not additive. This is also true for snapshots; avoid specifying metadata while creating a snapshot of an encrypted blob. If metadata must be set, be sure to call the `get_blob_metadata` method first to get the current encryption metadata, and avoid concurrent writes while metadata is being set.
- Enable the `require_encryption` flag on the service object for users that should work only with encrypted data. See below for more info.

The storage client library expects the provided KEK and key resolver to implement the following interface. [Azure Key Vault](#) support for Python KEK management is pending and will be integrated into this library when completed.

## Client API / Interface

After a storage service object (i.e. `blockblobservice`) has been created, the user may assign values to the fields that constitute an encryption policy: `key_encryption_key`, `key_resolver_function`, and `require_encryption`. Users can provide only a KEK, only a resolver, or both. `key_encryption_key` is the basic key type that is identified using a key identifier and that provides the logic for wrapping/unwrapping. `key_resolver_function` is used to resolve a key during the decryption process. It returns a valid KEK given a key identifier. This provides users the ability to choose between multiple keys that are managed in multiple locations.

The KEK must implement the following methods to successfully encrypt data:

- `wrap_key(cek)`: Wraps the specified CEK (bytes) using an algorithm of the user's choice. Returns the wrapped key.
- `get_key_wrap_algorithm()`: Returns the algorithm used to wrap keys.
- `get_kid()`: Returns the string key id for this KEK. The KEK must implement the following methods to successfully decrypt data:
- `unwrap_key(cek, algorithm)`: Returns the unwrapped form of the specified CEK using the string-specified algorithm.
- `get_kid()`: Returns a string key id for this KEK.

The key resolver must at least implement a method that, given a key id, returns the corresponding KEK implementing the interface above. Only this method is to be assigned to the `key_resolver_function` property on the service object.

- For encryption, the key is used always and the absence of a key will result in an error.
- For decryption:
  - The key resolver is invoked if specified to get the key. If the resolver is specified but does not have a mapping for the key identifier, an error is thrown.
  - If resolver is not specified but a key is specified, the key is used if its identifier matches the required key identifier. If the identifier does not match, an error is thrown.

The encryption samples in `azure.storage.samples` demonstrate a more detailed end-to-end scenario for blobs, queues and tables. Sample implementations of the KEK and key resolver are provided in

the sample files as KeyWrapper and KeyResolver respectively.

## RequireEncryption mode

Users can optionally enable a mode of operation where all uploads and downloads must be encrypted. In this mode, attempts to upload data without an encryption policy or download data that is not encrypted on the service will fail on the client. The `require_encryption` flag on the service object controls this behavior.

## Blob service encryption

Set the encryption policy fields on the `blockblobservice` object. Everything else will be handled by the client library internally.

```
Create the KEK used for encryption.
KeyWrapper is the provided sample implementation, but the user may use their own object as long as it
implements the interface above.
kek = KeyWrapper('local:key1') # Key identifier

Create the key resolver used for decryption.
KeyResolver is the provided sample implementation, but the user may use whatever implementation they choose
so long as the function set on the service object behaves appropriately.
key_resolver = KeyResolver()
key_resolver.put_key(kek)

Set the KEK and key resolver on the service object.
my_block_blob_service.key_encryption_key = kek
my_block_blob_service.key_resolver_funcion = key_resolver.resolve_key

Upload the encrypted contents to the blob.
my_block_blob_service.create_blob_from_stream(
 container_name, blob_name, stream)

Download and decrypt the encrypted contents from the blob.
blob = my_block_blob_service.get_blob_to_bytes(container_name, blob_name)
```

## Queue service encryption

Set the encryption policy fields on the `queueservice` object. Everything else will be handled by the client library internally.

```
Create the KEK used for encryption.
KeyWrapper is the provided sample implementation, but the user may use their own object as long as it
implements the interface above.
kek = KeyWrapper('local:key1') # Key identifier

Create the key resolver used for decryption.
KeyResolver is the provided sample implementation, but the user may use whatever implementation they choose
so long as the function set on the service object behaves appropriately.
key_resolver = KeyResolver()
key_resolver.put_key(kek)

Set the KEK and key resolver on the service object.
my_queue_service.key_encryption_key = kek
my_queue_service.key_resolver_funcion = key_resolver.resolve_key

Add message
my_queue_service.put_message(queue_name, content)

Retrieve message
retrieved_message_list = my_queue_service.get_messages(queue_name)
```

## Table service encryption

In addition to creating an encryption policy and setting it on request options, you must either specify an

`encryption_resolver_function` on the `tableservice`, or set the `encrypt` attribute on the `EntityProperty`.

## Using the resolver

```
Create the KEK used for encryption.
KeyWrapper is the provided sample implementation, but the user may use their own object as long as it
implements the interface above.
kek = KeyWrapper('local:key1') # Key identifier

Create the key resolver used for decryption.
KeyResolver is the provided sample implementation, but the user may use whatever implementation they choose
so long as the function set on the service object behaves appropriately.
key_resolver = KeyResolver()
key_resolver.put_key(kek)

Define the encryption resolver_function.

def my_encryption_resolver(pk, rk, property_name):
 if property_name == 'foo':
 return True
 return False

Set the KEK and key resolver on the service object.
my_table_service.key_encryption_key = kek
my_table_service.key_resolver_funcion = key_resolver.resolve_key
my_table_service.encryption_resolver_function = my_encryption_resolver

Insert Entity
my_table_service.insert_entity(table_name, entity)

Retrieve Entity
Note: No need to specify an encryption resolver for retrieve, but it is harmless to leave the property set.
my_table_service.get_entity(
 table_name, entity['PartitionKey'], entity['RowKey'])
```

## Using attributes

As mentioned above, a property may be marked for encryption by storing it in an `EntityProperty` object and setting the `encrypt` field.

```
encrypted_property_1 = EntityProperty(EdmType.STRING, value, encrypt=True)
```

## Encryption and performance

Note that encrypting your storage data results in additional performance overhead. The content key and IV must be generated, the content itself must be encrypted, and additional metadata must be formatted and uploaded. This overhead will vary depending on the quantity of data being encrypted. We recommend that customers always test their applications for performance during development.

## Next steps

- Download the [Azure Storage Client Library for Java PyPi package](#)
- Download the [Azure Storage Client Library for Python Source Code from GitHub](#)

# Require secure transfer to ensure secure connections

4/22/2020 • 2 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.

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, SMB 3.0 without encryption, or some versions of the Linux SMB client.

By default, the **Secure transfer required** property is enabled when you create a storage account.

## 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. And classic storage accounts are not supported.

## 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. Under **Secure transfer required**, select **Enabled**.

**Create storage account**

Basics Advanced Tags Review + create

**SECURITY**

Secure transfer required (i)  Disabled  Enabled

**VIRTUAL NETWORKS**

Allow access from  All networks  Selected network  
(i) All networks will be able to access this storage account. [Learn more](#)

**DATA LAKE STORAGE GEN2 (PREVIEW)**

Hierarchical namespace (i)  Disabled  Enabled

**Review + create** **Previous** **Next : Tags >**

### 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.

**requiresecurexfer - Configuration**

Storage account

Search (Ctrl+/  
)

**SETTINGS**

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Configuration (i)
- Shared access signature
- Properties
- Locks
- Automation script

**Save** **Discard**

The cost of your storage account depends on the usage and the options you choose below.

[Learn more](#)

Performance (i)  
 Standard  Premium

\* Secure transfer required (i)  
 Disabled  Enabled

Replication (i)  
 Read-access geo-redundant storage (RA-GRS)

### Require secure transfer from code

To require secure transfer programmatically, set the *supportsHttpsTrafficOnly* property 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 has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

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 a [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

4/10/2020 • 18 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 used. When network rules are configured, only applications requesting data over the specified set of networks can access a storage account. You can limit access to your storage account to requests originating from specified IP addresses, IP ranges or from a list of subnets in an Azure Virtual Network (VNet).

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 access 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.

## 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 has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

## 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 select 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 to 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 backup and restore VMs by creating an exception. This process is documented in the [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. To limit access to selected networks, you must first change the default action.

### WARNING

Making changes to network rules can impact your applications' ability to connect to Azure Storage. Setting the default network rule to **deny** blocks all access to the data unless specific network rules that **grant** access are also applied. Be sure to grant access to any allowed networks using network rules before you change the default rule to deny access.

### Managing default network access rules

You can manage default network access rules for storage accounts through the Azure portal, PowerShell, or CLIv2.

#### Azure portal

1. Go to the storage account you want to secure.
2. Click on the settings menu called **Firewalls and virtual networks**.
3. To deny access by default, choose to allow access from **Selected networks**. To allow traffic from all networks, choose to allow access from **All networks**.
4. Click **Save** to apply your changes.

#### PowerShell

1. Install the [Azure PowerShell](#) and [sign in](#).
2. Display the status of the default rule for the storage account.

```
(Get-AzStorageAccountNetworkRuleSet -ResourceGroupName "myresourcegroup" -AccountName "mystorageaccount").DefaultAction
```

3. Set the default rule to deny network access by default.

```
Update-AzStorageAccountNetworkRuleSet -ResourceGroupName "myresourcegroup" -Name "mystorageaccount" -DefaultAction Deny
```

- Set the default rule to allow network access by default.

```
Update-AzStorageAccountNetworkRuleSet -ResourceGroupName "myresourcegroup" -Name "mystorageaccount" -DefaultAction Allow
```

#### CLIV2

- Install the [Azure CLI](#) and [sign in](#).
- Display the status of the default rule for the storage account.

```
az storage account show --resource-group "myresourcegroup" --name "mystorageaccount" --query networkRuleSet.defaultAction
```

- Set the default rule to deny network access by default.

```
az storage account update --resource-group "myresourcegroup" --name "mystorageaccount" --default-action Deny
```

- Set the default rule to allow network access by default.

```
az storage account update --resource-group "myresourcegroup" --name "mystorageaccount" --default-action Allow
```

## 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.

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 100 virtual network rules, which may be combined with [IP network rules](#).

### Available virtual network regions

In general, service endpoints work between virtual networks and service instances in the same Azure region. When using service endpoints with Azure Storage, this scope grows to include the [paired region](#). 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.

#### NOTE

Service endpoints don't apply to traffic outside the region of the virtual network and the designated region pair. You can only apply network rules granting access from virtual networks to storage accounts in the primary region of a storage account or in the designated paired region.

## Required permissions

To apply a virtual network rule to a storage account, the user must have the appropriate permissions for the subnets being added. The permission needed is *Join Service to a Subnet* and is included in the *Storage Account Contributor* built-in role. It can also be added to custom role definitions.

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.

## Managing virtual network rules

You can manage virtual network rules for storage accounts through the Azure portal, PowerShell, or CLIV2.

### Azure portal

1. Go to the storage account you want to secure.
2. Click on the settings menu called **Firewalls and virtual networks**.
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**, click **Add existing virtual network**, select **Virtual networks** and **Subnets** options, and then click **Add**. To create a new virtual network and grant it access, click **Add new virtual network**. Provide the information necessary to create the new virtual network, and then click **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.

5. To remove a virtual network or subnet rule, click ... to open the context menu for the virtual network or subnet, and click **Remove**.
6. Click **Save** to apply your changes.

### PowerShell

1. Install the [Azure PowerShell](#) and sign in.

2. List virtual network rules.

```
(Get-AzStorageAccountNetworkRuleSet -ResourceGroupName "myresourcegroup" -AccountName
"mystorageaccount").VirtualNetworkRules
```

3. Enable service endpoint for Azure Storage on an existing virtual network and subnet.

```
Get-AzVirtualNetwork -ResourceGroupName "myresourcegroup" -Name "myvnet" | Set-AzVirtualNetworkSubnetConfig -Name "mysubnet" -AddressPrefix "10.0.0.0/24" -ServiceEndpoint "Microsoft.Storage" | Set-AzVirtualNetwork
```

#### 4. Add a network rule for a virtual network and subnet.

```
$subnet = Get-AzVirtualNetwork -ResourceGroupName "myresourcegroup" -Name "myvnet" | Get-AzVirtualNetworkSubnetConfig -Name "mysubnet"
Add-AzStorageAccountNetworkRule -ResourceGroupName "myresourcegroup" -Name "mystorageaccount" -VirtualNetworkResourceId $subnet.Id
```

#### TIP

To add a network rule for a subnet in a VNet belonging to another Azure AD tenant, use a fully-qualified **VirtualNetworkResourceId** parameter in the form "/subscriptions/subscription-ID/resourceGroups/resourceGroup-Name/providers/Microsoft.Network/virtualNetworks/vNet-name/subnets/subnet-name".

#### 5. Remove a network rule for a virtual network and subnet.

```
$subnet = Get-AzVirtualNetwork -ResourceGroupName "myresourcegroup" -Name "myvnet" | Get-AzVirtualNetworkSubnetConfig -Name "mysubnet"
Remove-AzStorageAccountNetworkRule -ResourceGroupName "myresourcegroup" -Name "mystorageaccount" -VirtualNetworkResourceId $subnet.Id
```

#### IMPORTANT

Be sure to [set the default rule to deny](#), or network rules have no effect.

#### CLIV2

##### 1. Install the [Azure CLI](#) and [sign in](#).

##### 2. List virtual network rules.

```
az storage account network-rule list --resource-group "myresourcegroup" --account-name "mystorageaccount" --query virtualNetworkRules
```

##### 3. Enable service endpoint for Azure Storage on an existing virtual network and subnet.

```
az network vnet subnet update --resource-group "myresourcegroup" --vnet-name "myvnet" --name "mysubnet" --service-endpoints "Microsoft.Storage"
```

##### 4. Add a network rule for a virtual network and subnet.

```
$subnetid=(az network vnet subnet show --resource-group "myresourcegroup" --vnet-name "myvnet" --name "mysubnet" --query id --output tsv)
az storage account network-rule add --resource-group "myresourcegroup" --account-name "mystorageaccount" --subnet $subnetid
```

**TIP**

To add a rule for a subnet in a VNet belonging to another Azure AD tenant, use a fully-qualified subnet ID in the form "/subscriptions/<subscription-ID>/resourceGroups/<resourceGroupName>/providers/Microsoft.Network/virtualNetworks/<vNet-name>/subnets/<subnet-name>".

You can use the **subscription** parameter to retrieve the subnet ID for a VNet belonging to another Azure AD tenant.

## 5. Remove a network rule for a virtual network and subnet.

```
$subnetid=(az network vnet subnet show --resource-group "myresourcegroup" --vnet-name "myvnet" --name "mysubnet" --query id --output tsv)
az storage account network-rule remove --resource-group "myresourcegroup" --account-name "mystorageaccount" --subnet $subnetid
```

**IMPORTANT**

Be sure to [set the default rule to deny](#), or network rules have no effect.

## Grant access from an internet IP range

You can configure storage accounts to allow access from specific public internet IP address ranges. This configuration grants access to specific internet-based services and on-premises networks and blocks general internet traffic.

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`.

**NOTE**

Small address ranges using "/31" or "/32" prefix sizes are not supported. These ranges should be configured using individual IP address rules.

IP network rules are only allowed 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.*`.

**NOTE**

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.

**NOTE**

Services deployed in the same region as the storage account use private Azure IP addresses for communication. Thus, you cannot restrict access to specific Azure services based on their public outbound IP address range.

Only IPV4 addresses are supported for configuration of storage firewall rules.

Each storage account supports up to 100 IP network rules.

## 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.

### Azure portal

1. Go to the storage account you want to secure.
2. Click on the settings menu called **Firewalls and virtual networks**.
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, click the trash can icon next to the address range.
6. Click **Save** to apply your changes.

### PowerShell

1. Install the [Azure PowerShell](#) and [sign in](#).

2. List IP network rules.

```
(Get-AzStorageAccountNetworkRuleSet -ResourceGroupName "myresourcegroup" -AccountName
"mystorageaccount").IPRules
```

3. Add a network rule for an individual IP address.

```
Add-AzStorageAccountNetworkRule -ResourceGroupName "myresourcegroup" -AccountName "mystorageaccount" -
IPAddressOrRange "16.17.18.19"
```

4. Add a network rule for an IP address range.

```
Add-AzStorageAccountNetworkRule -ResourceGroupName "myresourcegroup" -AccountName "mystorageaccount" -
IPAddressOrRange "16.17.18.0/24"
```

5. Remove a network rule for an individual IP address.

```
Remove-AzStorageAccountNetworkRule -ResourceGroupName "myresourcegroup" -AccountName "mystorageaccount"
-IPAddressOrRange "16.17.18.19"
```

6. Remove a network rule for an IP address range.

```
Remove-AzStorageAccountNetworkRule -ResourceGroupName "myresourcegroup" -AccountName "mystorageaccount"
-IPAddressOrRange "16.17.18.0/24"
```

#### IMPORTANT

Be sure to [set the default rule to deny](#), or network rules have no effect.

#### CLIV2

1. Install the [Azure CLI](#) and [sign in](#).

2. List IP network rules.

```
az storage account network-rule list --resource-group "myresourcegroup" --account-name
"mystorageaccount" --query ipRules
```

3. Add a network rule for an individual IP address.

```
az storage account network-rule add --resource-group "myresourcegroup" --account-name
"mystorageaccount" --ip-address "16.17.18.19"
```

4. Add a network rule for an IP address range.

```
az storage account network-rule add --resource-group "myresourcegroup" --account-name
"mystorageaccount" --ip-address "16.17.18.0/24"
```

5. Remove a network rule for an individual IP address.

```
az storage account network-rule remove --resource-group "myresourcegroup" --account-name
"mystorageaccount" --ip-address "16.17.18.19"
```

6. Remove a network rule for an IP address range.

```
az storage account network-rule remove --resource-group "myresourcegroup" --account-name
"mystorageaccount" --ip-address "16.17.18.0/24"
```

#### IMPORTANT

Be sure to [set the default rule to deny](#), or network rules have no effect.

## Exceptions

Network rules help to create a secure environment for connections between your applications and your data for most scenarios. However, some applications depend on Azure services that cannot be uniquely isolated through virtual network or IP address rules. But such services must be granted to storage to enable full application functionality. In such situations, you can use the ***Allow trusted Microsoft services...*** setting to enable such services to access your data, logs, or analytics.

### Trusted Microsoft services

Some Microsoft services operate from networks that can't be included in your network rules. You can grant a subset of such trusted Microsoft services access to the storage account, while maintaining network rules for other

apps. These trusted services will then use strong authentication to connect to your storage account securely. We've enabled two modes of trusted access for Microsoft services.

- 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.
- Resources of some services can be granted explicit access to your storage account by **assigning an RBAC role** to its system-assigned managed identity.

When you enable the **Allow trusted Microsoft services...** setting, resources of the following services that are registered in the same subscription as your storage account are granted access for a limited set of operations as described:

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> .
Azure Import Export	Microsoft.ImportExport	Enables import of data to Azure and export of data from Azure using Import/Export service. <a href="#">Learn more</a> .
Azure Monitor	Microsoft.Insights	Allows writing of monitoring data to a secured storage account, including resource diagnostic 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. <a href="#">Learn more</a> .

Service	Resource provider name	Operations allowed
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> .

The **Allow trusted Microsoft services...** setting also allows a particular instance of the below services to access the storage account, if you explicitly [assign an RBAC role](#) to the [system-assigned managed identity](#) for that resource instance. In this case, the scope of access for the instance corresponds to the RBAC role assigned to the managed identity.

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 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 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 SQL Data Warehouse	Microsoft.Sql	Allows import and export of data from specific SQL Database instances using PolyBase. <a href="#">Learn more</a> .
Azure Stream Analytics	Microsoft.StreamAnalytics	Allows data from a streaming job to be written to Blob storage. This feature is currently in preview. <a href="#">Learn more</a> .
Azure Synapse Analytics	Microsoft.Synapse/workspaces	Enables access to data in Azure Storage from Synapse Analytics.

### Storage analytics data access

In some cases, access to read diagnostic 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. [Learn more about working with storage analytics](#).

### Managing exceptions

You can manage network rule exceptions through the Azure portal, PowerShell, or Azure CLI v2.

#### Azure portal

1. Go to the storage account you want to secure.

2. Click on the settings menu called **Firewalls and virtual networks**.
3. Check that you've selected to allow access from **Selected networks**.
4. Under **Exceptions**, select the exceptions you wish to grant.
5. Click **Save** to apply your changes.

#### PowerShell

1. Install the [Azure PowerShell](#) and [sign in](#).

2. Display the exceptions for the storage account network rules.

```
(Get-AzStorageAccountNetworkRuleSet -ResourceGroupName "myresourcegroup" -Name "mystorageaccount").Bypass
```

3. Configure the exceptions to the storage account network rules.

```
Update-AzStorageAccountNetworkRuleSet -ResourceGroupName "myresourcegroup" -Name "mystorageaccount" -Bypass AzureServices,Metrics,Logging
```

4. Remove the exceptions to the storage account network rules.

```
Update-AzStorageAccountNetworkRuleSet -ResourceGroupName "myresourcegroup" -Name "mystorageaccount" -Bypass None
```

#### IMPORTANT

Be sure to [set the default rule to deny](#), or removing exceptions have no effect.

#### CLIv2

1. Install the [Azure CLI](#) and [sign in](#).

2. Display the exceptions for the storage account network rules.

```
az storage account show --resource-group "myresourcegroup" --name "mystorageaccount" --query networkRuleSet.bypass
```

3. Configure the exceptions to the storage account network rules.

```
az storage account update --resource-group "myresourcegroup" --name "mystorageaccount" --bypass Logging Metrics AzureServices
```

4. Remove the exceptions to the storage account network rules.

```
az storage account update --resource-group "myresourcegroup" --name "mystorageaccount" --bypass None
```

#### IMPORTANT

Be sure to [set the default rule to deny](#), or removing exceptions have no effect.

## Next steps

Learn more about Azure Network service endpoints in [Service endpoints](#).

Dig deeper into Azure Storage security in [Azure Storage security guide](#).

# Enable secure TLS for Azure Storage client

12/23/2019 • 2 minutes to read • [Edit Online](#)

Transport Layer Security (TLS) and Secure Sockets Layer (SSL) are cryptographic protocols that provide communications security over a computer network. SSL 1.0, 2.0 and 3.0 have been found to be vulnerable. They have been prohibited by RFC. TLS 1.0 becomes insecure for using insecure Block cipher (DES CBC and RC2 CBC) and Stream cipher (RC4). PCI council also suggested the migration to higher TLS versions. For more details, you can see [Transport Layer Security \(TLS\)](#).

Azure Storage has stopped SSL 3.0 since 2015 and uses TLS 1.2 on public HTTPs endpoints but TLS 1.0 and TLS 1.1 are still supported for backward compatibility.

In order to ensure secure and compliant connection to Azure Storage, you need to enable TLS 1.2 or newer version in client side before sending requests to operate Azure Storage service.

## Enable TLS 1.2 in .NET client

For the client to negotiate TLS 1.2, the OS and the .NET Framework version both need to support TLS 1.2. See more details in [Support for TLS 1.2](#).

The following sample shows how to enable TLS 1.2 in your .NET client.

```
static void EnableTls12()
{
 // Enable TLS 1.2 before connecting to Azure Storage
 System.Net.ServicePointManager.SecurityProtocol = System.Net.SecurityProtocolType.Tls12;

 // Connect to Azure Storage
 CloudStorageAccount storageAccount =
 CloudStorageAccount.Parse("DefaultEndpointsProtocol=https;AccountName={yourstorageaccount};AccountKey=
 {yourstorageaccountkey};EndpointSuffix=core.windows.net");
 CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();

 CloudBlobContainer container = blobClient.GetContainerReference("foo");
 container.CreateIfNotExists();
}
```

## Enable TLS 1.2 in PowerShell client

### NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

The following sample shows how to enable TLS 1.2 in your PowerShell client.

```

Enable TLS 1.2 before connecting to Azure Storage
[System.Net.ServicePointManager]::SecurityProtocol = [System.Net.SecurityProtocolType]::Tls12;

$resourceGroup = "{YourResourceGroupName}"
$storageAccountName = "{YourStorageAccountName}"
$prefix = "foo"

Connect to Azure Storage
$storageAccount = Get-AzStorageAccount -ResourceGroupName $resourceGroup -Name $storageAccountName
$cxt = $storageAccount.Context
$listOfContainers = Get-AzStorageContainer -Context $cxt -Prefix $prefix
$listOfContainers

```

## Verify TLS 1.2 connection

You can use Fiddler to verify if TLS 1.2 is used actually. Open Fiddler to start capturing client network traffic then execute above sample. Then you can find the TLS version in the connection that the sample makes.

The following screenshot is a sample for the verification.

A screenshot of the Fiddler Web Debugger interface. The main window shows three captured sessions. Session 1 is a response from config.edge.skype.com:443. Session 2 is a response from client-office365-test.medge.net:443. Session 3 is a response from blob.core.windows.net:443. The details pane for session 3 is expanded, showing a ClientHello message. In the 'Version' field, the value '3.3 (TLS 1.2)' is highlighted with a red box. Below it, the message continues with random bytes, timestamp, session ID, extensions (server\_name, elliptic\_curves, ec\_point\_formats, signature\_algs), and ciphers (TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_GCM\_SHA384, TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256, etc.).

## See also

- [Transport Layer Security \(TLS\)](#)
- [PCI compliance on TLS](#)
- [Enable TLS in Java client](#)

# Use .NET to manage directories, files, and ACLs in Azure Data Lake Storage Gen2

3/20/2020 • 5 minutes to read • [Edit Online](#)

This article shows you how to use .NET to create and manage directories, files, and permissions in storage accounts that has hierarchical namespace (HNS) enabled.

[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.

## 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.Storage.Files.DataLake;
using Azure.Storage.Files.DataLake.Models;
using Azure.Storage;
using System.IO;
using Azure;
```

## 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 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 (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 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 file system

A file system acts as a container for your files. You can create one by calling the [DataLakeServiceClient.CreateFileSystem](#) method.

This example creates a file system 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 file system, 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();
}
```

## Manage a directory ACL

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.

### 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](#).

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("my-directory");

 PathAccessControl directoryAccessControl =
 await directoryClient.GetAccessControlAsync();

 Console.WriteLine(directoryAccessControl.AccessControlList);

 IList<PathAccessControlItem> accessControlList
 = PathAccessControlExtensions.ParseAccessControlList
 ("user::rwx,group::r-x,other::rw-");

 directoryClient.SetAccessControlList(accessControlList);
}

```

## 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:\\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:\\file-to-upload.txt");

 await fileClient.UploadAsync(fileStream);

}

```

## Manage a file ACL

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.

### 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](#).

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();

 Console.WriteLine(FileAccessControl.AccessControlList);

 IList<PathAccessControlItem> accessControlList
 = PathAccessControlExtensions.ParseAccessControlList
 ("user::rwx,group::r-x,other::rw-");

 fileClient.SetAccessControlList(accessControlList);
}

```

## 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:\\\\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, files, and ACLs in Azure Data Lake Storage Gen2

3/20/2020 • 5 minutes to read • [Edit Online](#)

This article shows you how to use Java to create and manage directories, files, and permissions in storage accounts that has hierarchical namespace (HNS) enabled.

[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.

## 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.core.credential.TokenCredential;
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.PathAccessControl;
import com.azure.storage.file.datalake.models.PathAccessControlEntry;
import com.azure.storage.file.datalake.models.PathItem;
import com.azure.storage.file.datalake.models.PathPermissions;
import com.azure.storage.file.datalake.models.RolePermissions;
```

## 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 file system

A file system acts as a container for your files. You can create one by calling the **DataLakeServiceClient.createFileSystem** method.

This example creates a file system named `my-file-system`.

```

static 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 file system, and then adds a sub-directory named `my-subdirectory`.

```
static 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`.

```
static 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`.

```
static 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`.

```
static public void DeleteDirectory(DataLakeFileSystemClient fileSystemClient){

 DataLakeDirectoryClient directoryClient =
 fileSystemClient.getDirectoryClient("my-directory");

 directoryClient.deleteWithResponse(true, null, null, null);
}
```

## Manage 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](#).

```
static public void ManageDirectoryACLs(DataLakeFileSystemClient fileSystemClient){

 DataLakeDirectoryClient directoryClient =
 fileSystemClient.getDirectoryClient("my-directory");

 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));
}
```

## 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`:

```
static 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:\\\\mytestfile.txt");

 InputStream targetStream = 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.

```
static public void UploadFileBulk(DataLakeFileSystemClient fileSystemClient)
 throws FileNotFoundException{

 DataLakeDirectoryClient directoryClient =
 fileSystemClient.getDirectoryClient("my-directory");

 DataLakeFileClient fileClient = directoryClient.getFileClient("uploaded-file.txt");

 fileClient.uploadFromFile("C:\\\\mytestfile.txt");

}
```

## Manage 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](#).

```
static 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));

}
```

## 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 .NET file processing API to save bytes from the stream to a file.

```
static 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:\\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`.

```
static 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, files, and ACLs in Azure Data Lake Storage Gen2

4/12/2020 • 5 minutes to read • [Edit Online](#)

This article shows you how to use Python to create and manage directories, files, and permissions in storage accounts that has hierarchical namespace (HNS) enabled.

[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 (HNS) 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.

```
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 (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 file system

A file system acts as a container for your files. You can create one by calling the **FileSystemDataLakeServiceClient.create\_file\_system** method.

This example creates a file system 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 file system.

```
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-subdirectory-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(rename_destination=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)
```

## Manage directory permissions

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.

### 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](#).

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)

```

## 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)

```

## Manage file permissions

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.

### 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](#).

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)

```

## 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 to manage directories, files, and ACLs in Azure Data Lake Storage Gen2

3/20/2020 • 5 minutes to read • [Edit Online](#)

This article shows you how to use JavaScript to create and manage directories, files, and permissions in storage accounts that has hierarchical namespace (HNS) enabled.

[Package \(Node Package Manager\)](#) | [Samples](#) | [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.
- 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 = 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 (AD).

## Connect by using Azure Active Directory (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 file system

A file system acts as a container for your files. You can create one by getting a **FileSystemClient** instance, and then calling the **FileSystemClient.Create** method.

This example creates a file system 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 file system.

```
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();

}
```

## Manage 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);
}
```

## 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);

}
```

## Manage 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);
}

```

## 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](#)

# Use PowerShell to manage directories, files, and ACLs in Azure Data Lake Storage Gen2

4/21/2020 • 9 minutes to read • [Edit Online](#)

This article shows you how to use PowerShell to create and manage directories, files, and permissions in storage accounts that has hierarchical namespace (HNS) enabled.

[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.
- .NET Framework is 4.7.2 or greater installed. 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

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
```

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>
```

Next, choose how you want your commands to obtain authorization to the storage account.

### Option 1: Obtain authorization by using Azure Active Directory (AD)

With this approach, the system ensures that your user account has the appropriate role-based access control

(RBAC) assignments and ACL permissions.

```
$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 RBAC or ACL permissions.

```
$storageAccount = Get-AzStorageAccount -ResourceGroupName "<resource-group-name>" -AccountName "<storage-account-name>"
$ctx = $storageAccount.Context
```

## Create a file system

A file system acts as a container for your files. You can create one by using the `New-AzDatalakeGen2FileSystem` cmdlet.

This example creates a file system named `my-file-system`.

```
$filesystemName = "my-file-system"
New-AzDatalakeGen2FileSystem -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 file system.

```
$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
```

## 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 $dirname1 -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
```

To list the contents of a file system, omit the `-Path` parameter from the command.

## 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
```

## 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.

## Manage access permissions

You can get, set, and update access permissions of file systems, directories and files. These permissions are captured in access control lists (ACLs).

### NOTE

If you're using Azure Active Directory (Azure AD) to authorize commands, 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 in Azure Data Lake Storage Gen2](#).

### Get an ACL

Get the ACL of a directory or file by using the `Get-AzDataLakeGen2Item` cmdlet.

This example gets the ACL of a **file system** 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 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 a **file system** 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
```

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
```

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 on all items in a file system

You can use the `Get-AzDataLakeGen2Item` and the `-Recurse` parameter together with the `Update-AzDataLakeGen2Item` cmdlet to recursively set the ACL of all directories and files in a file system.

```
$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
Get-AzDataLakeGen2ChildItem -Context $ctx -FileSystem $filesystemName -Recurse | Update-AzDataLakeGen2Item -
Acl $acl
```

### Add or update an ACL entry

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
```

### 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
```

## 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 Get-AzDataLakeGen2ChildItem cmdlet only lists the first level child items. The -Recurse parameter lists child items recursively.
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](#)
- [Using Azure PowerShell with Azure Storage](#).
- [Storage PowerShell cmdlets](#).

# Use Azure CLI to manage directories, files, and ACLs in Azure Data Lake Storage Gen2 (preview)

2/20/2020 • 7 minutes to read • [Edit Online](#)

This article shows you how to use the [Azure Command-Line Interface \(CLI\)](#) to create and manage directories, files, and permissions in storage accounts that have a hierarchical namespace.

## IMPORTANT

The `storage-preview` extension that is featured in this article is currently in public preview.

[Sample](#) | [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 `2.0.67` or higher.

## Install the storage CLI extension

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 `2.0.67` or higher by using the following command.

```
az --version
```

If your version of Azure CLI is lower than `2.0.67`, then install a later version. See [Install the Azure CLI](#).

3. Install the `storage-preview` extension.

```
az extension add -n storage-preview
```

## 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 [Sign in 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.

## Create a file system

A file system acts as a container for your files. You can create one by using the `az storage container create` command.

This example creates a file system named `my-file-system`.

```
az storage container create --name my-file-system --account-name mystorageaccount
```

## Create a directory

Create a directory reference by using the `az storage blob directory create` command.

This example adds a directory named `my-directory` to a file system named `my-file-system` that is located in an account named `mystorageaccount`.

```
az storage blob directory create -c my-file-system -d my-directory --account-name mystorageaccount
```

## Show directory properties

You can print the properties of a directory to the console by using the `az storage blob show` command.

```
az storage blob directory show -c my-file-system -d my-directory --account-name mystorageaccount
```

## Rename or move a directory

Rename or move a directory by using the `az storage blob directory move` command.

This example renames a directory from the name `my-directory` to the name `my-new-directory`.

```
az storage blob directory move -c my-file-system -d my-new-directory -s my-directory --account-name mystorageaccount
```

## Delete a directory

Delete a directory by using the `az storage blob directory delete` command.

This example deletes a directory named `my-directory`.

```
az storage blob directory delete -c my-file-system -d my-directory --account-name mystorageaccount
```

## Check if a directory exists

Determine if a specific directory exists in the file system by using the `az storage blob directory exist` command.

This example reveals whether a directory named `my-directory` exists in the `my-file-system` file system.

```
az storage blob directory exists -c my-file-system -d my-directory --account-name mystorageaccount
```

## Download from a directory

Download a file from a directory by using the `az storage blob directory download` command.

This example downloads a file named `upload.txt` from a directory named `my-directory`.

```
az storage blob directory download -c my-file-system --account-name mystorageaccount -s "my-directory/upload.txt" -d "C:\mylocalfolder\download.txt"
```

This example downloads an entire directory.

```
az storage blob directory download -c my-file-system --account-name mystorageaccount -s "my-directory/" -d "C:\mylocalfolder" --recursive
```

## List directory contents

List the contents of a directory by using the `az storage blob directory list` command.

This example lists the contents of a directory named `my-directory` that is located in the `my-file-system` file system of a storage account named `mystorageaccount`.

```
az storage blob directory list -c my-file-system -d my-directory --account-name mystorageaccount
```

## Upload a file to a directory

Upload a file to a directory by using the `az storage blob directory upload` command.

This example uploads a file named `upload.txt` to a directory named `my-directory`.

```
az storage blob directory upload -c my-file-system --account-name mystorageaccount -s "C:\mylocaldirectory\upload.txt" -d my-directory
```

This example uploads an entire directory.

```
az storage blob directory upload -c my-file-system --account-name mystorageaccount -s "C:\mylocaldirectory\" -d my-directory --recursive
```

## Show file properties

You can print the properties of a file to the console by using the `az storage blob show` command.

```
az storage blob show -c my-file-system -b my-file.txt --account-name mystorageaccount
```

## Rename or move a file

Rename or move a file by using the `az storage blob move` command.

This example renames a file from the name `my-file.txt` to the name `my-file-renamed.txt`.

```
az storage blob move -c my-file-system -d my-file-renamed.txt -s my-file.txt --account-name mystorageaccount
```

## Delete a file

Delete a file by using the `az storage blob delete` command.

This example deletes a file named `my-file.txt`

```
az storage blob delete -c my-file-system -b my-file.txt --account-name mystorageaccount
```

## Manage permissions

You can get, set, and update access permissions of directories and files.

### NOTE

If you're using Azure Active Directory (Azure AD) to authorize commands, 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 in Azure Data Lake Storage Gen2](#).

### Get directory and file permissions

Get the ACL of a directory by using the `az storage blob directory access show` command.

This example gets the ACL of a directory, and then prints the ACL to the console.

```
az storage blob directory access show -d my-directory -c my-file-system --account-name mystorageaccount
```

Get the access permissions of a file by using the `az storage blob access show` command.

This example gets the ACL of a file and then prints the ACL to the console.

```
az storage blob access show -b my-directory/upload.txt -c my-file-system --account-name mystorageaccount
```

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 directory and file permissions

Use the `az storage blob directory 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 blob directory access set -a "user::rw-,group::rw-,other::wx" -d my-directory -c my-file-system --account-name mystorageaccount
```

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 blob directory access set -a "default:user::rw-,group::rw-,other::wx" -d my-directory -c my-file-system --account-name mystorageaccount
```

Use the `az storage blob 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 blob access set -a "user::rw-,group::rw-,other::wx" -b my-directory/upload.txt -c my-file-system --account-name mystorageaccount
```

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](#).

## Update directory and file permissions

Another way to set this permission is to use the `az storage blob directory access update` or `az storage blob access update` 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 blob directory access update --permissions "rwxrwxrwx" -d my-directory -c my-file-system --account-name mystorageaccount
```

This example updates the ACL of a file.

```
az storage blob access update --permissions "rwxrwxrwx" -b my-directory/upload.txt -c my-file-system --account-name mystorageaccount
```

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 blob directory access update --owner xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx -d my-directory -c my-file-system --account-name mystorageaccount
```

This example changes the owner of a file.

```
az storage blob access update --owner xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx -b my-directory/upload.txt -c my-file-system --account-name mystorageaccount
```

## Manage user-defined metadata

You can add user-defined metadata to a file or directory by using the `az storage blob directory metadata update` command with one or more name-value pairs.

This example adds user-defined metadata for a directory named `my-directory` directory.

```
az storage blob directory metadata update --metadata tag1=value1 tag2=value2 -c my-file-system -d my-directory --account-name mystorageaccount
```

This example shows all user-defined metadata for directory named `my-directory`.

```
az storage blob directory metadata show -c my-file-system -d my-directory --account-name mystorageaccount
```

## See also

- [Sample](#)
- [Gen1 to Gen2 mapping](#)
- [Give feedback](#)
- [Known issues](#)
- [Source code](#)

# Use Azure Storage Explorer to manage directories, files, and ACLs in Azure Data Lake Storage Gen2

1/24/2020 • 4 minutes to read • [Edit Online](#)

This article shows you how to use [Azure Storage Explorer](#) to create and manage directories, files, and permissions in storage accounts that has hierarchical namespace (HNS) enabled.

## Prerequisites

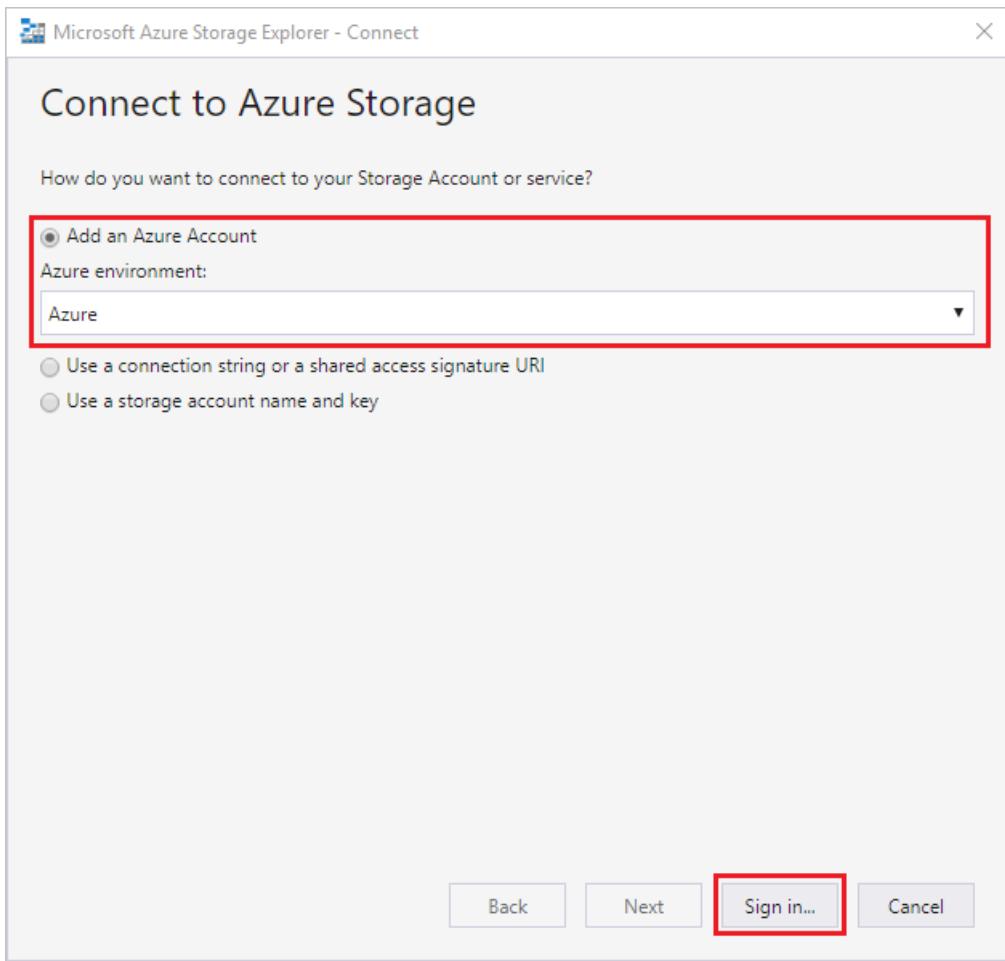
- An Azure subscription. See [Get Azure free trial](#).
- A storage account that has hierarchical namespace (HNS) enabled. Follow [these](#) instructions to create one.
- Azure Storage Explorer installed on your local computer. To install Azure Storage Explorer for Windows, Macintosh, or Linux, see [Azure Storage Explorer](#).

## Sign in to Storage Explorer

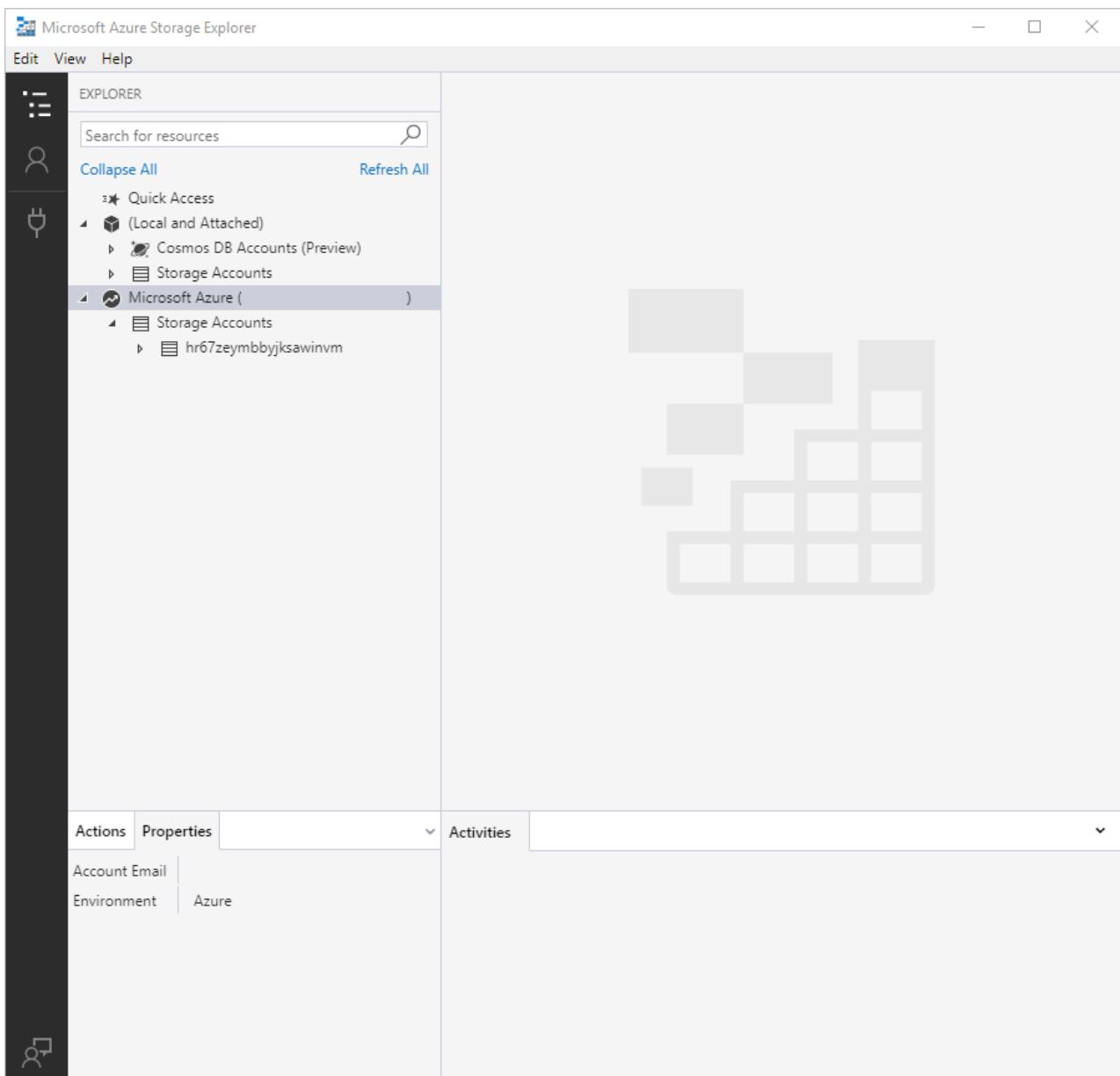
When you first start Storage Explorer, the **Microsoft Azure Storage Explorer - Connect** window appears. While Storage Explorer provides several ways to connect to storage accounts, only one way is currently supported for managing ACLs.

TASK	PURPOSE
Add an Azure Account	Redirects you to your organization's sign-in page to authenticate you to Azure. Currently this is the only supported authentication method if you want to manage and set ACLs.
Use a connection string or shared access signature URI	Can be used to directly access a container or storage account with a SAS token or a shared connection string.
Use a storage account name and key	Use the storage account name and key of your storage account to connect to Azure storage.

Select **Add an Azure Account** and click **Sign in...**. Follow the on-screen prompts to sign into your Azure account.

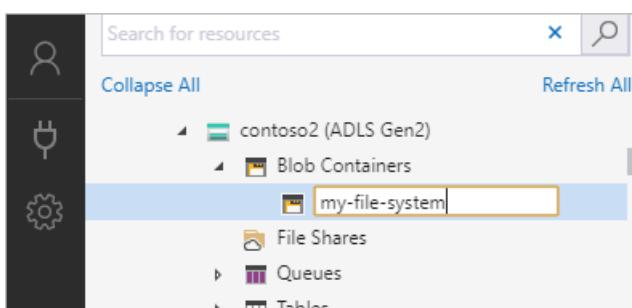


When it completes connecting, Azure Storage Explorer loads with the **Explorer** tab shown. This view gives you insight to all of your Azure storage accounts as well as local storage configured through the [Azure Storage Emulator](#), [Cosmos DB](#) accounts, or [Azure Stack](#) environments.



## Create a container

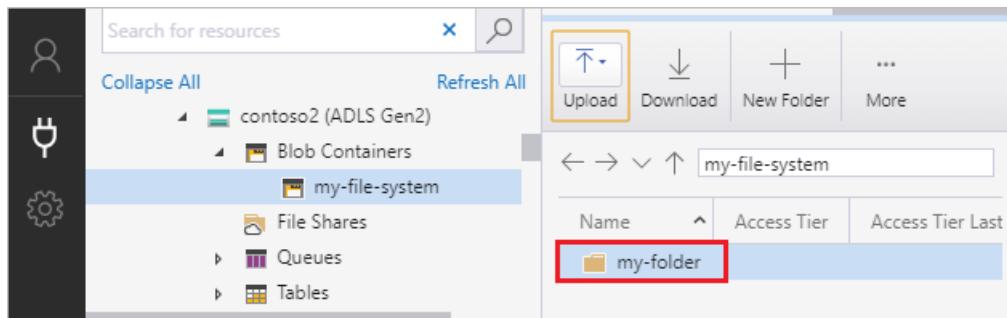
A container holds directories and files. To create one, expand the storage account you created in the proceeding step. Select **Blob Containers**, right-click and select **Create Blob Container**. Enter the name for your container. See the [Create a container](#) section for a list of rules and restrictions on naming containers. When complete, press **Enter** to create the container. Once 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.

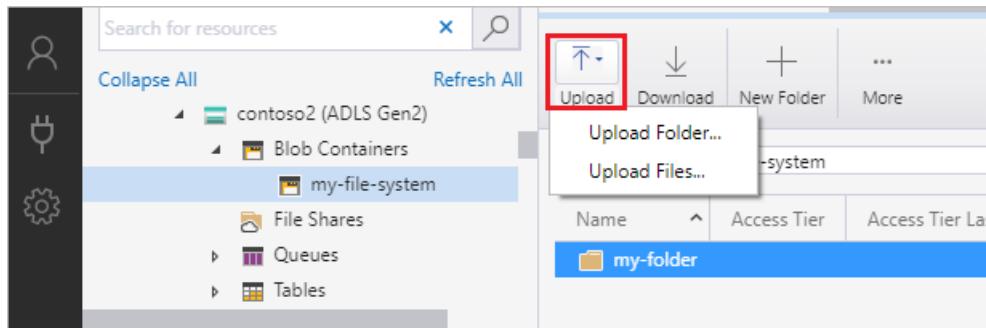
Once the directory has been successfully created, it appears in the editor window.



## Upload blobs to the directory

On the directory ribbon, chose the **Upload** button. This operation gives you the option to upload a folder or a file.

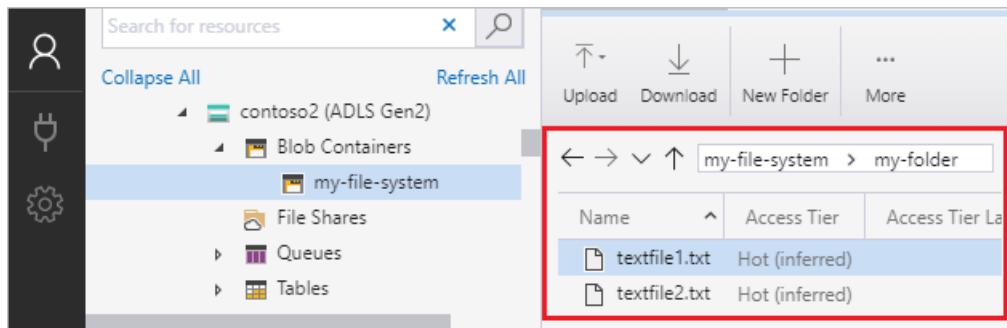
Choose the files or folder to upload.



When you select **OK**, 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 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 **Save** to start the download of a file to the local location.

## Managing access

You can set permissions at the root of your container. To do so, you must be logged into Azure Storage Explorer with your individual account with rights to do so (as opposed to with a connection string). Right-click your container and select **Manage Permissions**, bringing up the **Manage Permission** dialog box.

## Manage Permissions

Managing permissions for: examplefilesystem/

Users and groups:

- \$superuser (Owner)
- \$superuser (Owning Group)
- Other
- Mask

### Permissions for: \$superuser

Read    Write    Execute

<input checked="" type="checkbox"/> Access	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/> Default *	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

\* This will automatically add these permissions to all new children of this directory.  
[Learn more about default ACLs.](#)

Add user or group:

 Add

Save Cancel

The **Manage Permission** 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.

To add a new user or group to the access control list, select the **Add user or group** field.

Enter the corresponding Azure Active Directory (AAD) 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 AAD and maintain permissions on the group rather than individual users. For details on this recommendation, as well as other best practices, see [best practices for Data Lake Storage Gen2](#).

There are two categories of permissions you can assign: access ACLs and default ACLs.

- **Access:** Access ACLs control access to an object. Files and directories both have access ACLs.
- **Default:** A template of ACLs associated with a directory that determines the access ACLs for any child items

that are created under that directory. Files do not have default ACLs.

Within both of these categories, there are three permissions you can then assign on files or directories: **Read**, **Write**, and **Execute**.

**NOTE**

Making selections here will not set permissions on any currently existing item inside the directory. You must go to each individual item and set the permissions manually, if the file already exists.

You can manage permissions on individual directories, as well as individual files, which are what allows you fine grained access control. The process for managing permissions for both directories and files is the same as described above. Right-click the file or directory you wish to manage permissions on and follow the same process.

## Next steps

Learn access control lists in Data Lake Storage Gen2.

[Access control in Azure Data Lake Storage Gen2](#)

# Using the HDFS CLI with Data Lake Storage Gen2

8/23/2019 • 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@clustername-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](#)

# Filter data by using Azure Data Lake Storage query acceleration (preview)

4/21/2020 • 6 minutes to read • [Edit Online](#)

This article shows you how to use query acceleration (preview) to retrieve a subset of data from your storage account.

Query acceleration (preview) is a new capability for Azure Data Lake Storage that 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 \(preview\)](#).

## NOTE

The query acceleration feature is in public preview, and is available in the Canada Central and France Central regions. To review limitations, see the [Known issues](#) article. To enroll in the preview, see [this form](#).

## Prerequisites

- [.NET](#)
- [Java](#)
- 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](#).
- [.NET SDK](#).

## Install packages

- [.NET](#)
- [Java](#)

1. Download the query acceleration packages. You can obtain a compressed .zip file that contains these packages by using this link: <https://aka.ms/adls/qqsdk/.net>.
2. Extract the contents of this file to your project directory.
3. Open your project file (.csproj) in a text editor, and add these package references inside of the <Project> element.

```
<ItemGroup>
 <PackageReference Include="Azure.Storage.Blobs" Version="12.5.0-preview.1" />
 <PackageReference Include="Azure.Storage.Common" Version="12.4.0-preview.1" />
 <PackageReference Include="Azure.Storage.QuickQuery" Version="12.0.0-preview.1" />
</ItemGroup>
```

4. Restore the preview SDK packages. This example command restores the preview SDK packages by using the `dotnet restore` command.

```
dotnet restore --source C:\Users\contoso\myProject
```

5. Restore all other dependencies from the public NuGet repository.

```
dotnet restore
```

## Add statements

- [.NET](#)
- [Java](#)

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;
using Azure.Storage.QuickQuery;
using Azure.Storage.QuickQuery.Models;
```

Query acceleration retrieves CSV and Json formatted data. Therefore, make sure to add `using` statements for any CSV or Json parsing libraries that you choose to use. The examples that appear in this article parse a CSV file by using the [CsvHelper](#) library that is available on NuGet. Therefore, we'd add these `using` statements to the top of the code file.

```
using CsvHelper;
using CsvHelper.Configuration;
```

To compile examples presented in this article, you'll also need to add these `using` statements as well.

```
using System.Threading.Tasks;
using System.IO;
using System.Globalization;
using System.Threading;
using System.Linq;
```

## 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.

- [.NET](#)
- [Java](#)

The `async` method `BlobQuickQueryClient.QueryAsync` sends the query to the query acceleration API, and then streams the results back to the application as a [Stream](#) object.

```

static async Task QueryHemingway(BlockBlobClient blob)
{
 string query = @"SELECT * FROM BlobStorage WHERE _3 = 'Hemingway, Ernest'";
 await DumpQueryCsv(blob, query, false);
}

private static async Task DumpQueryCsv(BlockBlobClient blob, string query, bool headers)
{
 try
 {
 using (var reader = new StreamReader((await blob.GetQuickQueryClient().QueryAsync(query,
 new CsvTextConfiguration() { HasHeaders = headers },
 new CsvTextConfiguration() { HasHeaders = false },
 new ErrorHandler(),
 new BlobRequestConditions(),
 new ProgressHandler(),
 CancellationToken.None)).Value.Content))
 {
 using (var parser = new CsvReader(reader, new CsvConfiguration(CultureInfo.CurrentCulture)
 { HasHeaderRecord = false }))
 {
 while (await parser.ReadAsync())
 {
 parser.Context.Record.All(cell =>
 {
 Console.Out.Write(cell + " ");
 return true;
 });
 Console.Out.WriteLine();
 }
 }
 }
 }
 catch (Exception ex)
 {
 Console.Error.WriteLine("Exception: " + ex.ToString());
 }
}

class ErrorHandler : IBlobQueryErrorReceiver
{
 public void ReportError(BlobQueryError err)
 {
 Console.Error.WriteLine(String.Format("Error: {1}:{2}", err.Name, err.Description));
 }
}

class ProgressHandler : IProgress<long>
{
 public void Report(long value)
 {
 Console.Error.WriteLine("Bytes scanned: " + value.ToString());
 }
}

```

## 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.

This code retrieves only the `PublicationYear` 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.

- [.NET](#)
- [Java](#)

```
static async Task QueryPublishDates(BlockBlobClient blob)
{
 string query = @"SELECT PublicationYear FROM BlobStorage";
 await DumpQueryCsv(blob, query, true);
}
```

The following code combines row filtering and column projections into the same query.

- [.NET](#)
- [Java](#)

```
static async Task QueryMysteryBooks(BlockBlobClient blob)
{
 string query = @"SELECT BibNum, Title, Author, ISBN, Publisher FROM BlobStorage WHERE Subjects LIKE
'%Mystery%'";
 await DumpQueryCsv(blob, query, true);
}
```

## Next steps

- [Query acceleration enrollment form](#)
- [Azure Data Lake Storage query acceleration \(preview\)](#)
- [Query acceleration SQL language reference \(preview\)](#)
- [Query acceleration REST API reference](#)

# Get started with AzCopy

4/6/2020 • 11 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 files.

## 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 \(tar\)](#)
- [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.

## NOTE

If you want to copy data to and from your [Azure Table storage service](#), then install [AzCopy version 7.3](#).

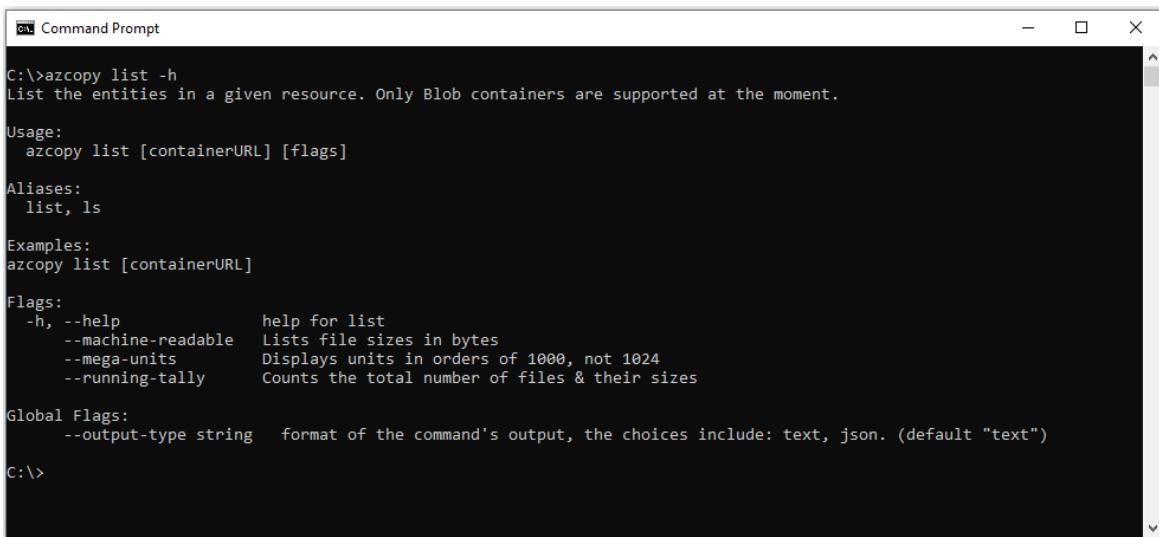
## Run AzCopy

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.

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`).



```
C:\>azcopy list -h
List the entities in a given resource. Only Blob containers are supported at the moment.

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 & their sizes

Global Flags:
 --output-type string format of the command's output, the choices include: text, json. (default "text")

C:\>
```

To find detailed reference documentation for each command and command parameter, see [azcopy](#)

**NOTE**

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.

## Choose how you'll provide authorization credentials

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

By using Azure Active Directory, you can provide credentials once instead of having to append a SAS token to each command.

**NOTE**

In the current release, if you plan to copy blobs between storage accounts, you'll have to append a SAS token to each source URL. You can omit the SAS token only from the destination URL. For examples, see [Copy blobs between storage accounts](#).

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](#) has been assigned to your user identity, managed identity, or service principal.

User identities, managed identities, and service principals are each a type of *security principal*, so we'll use the term *security principal* for the remainder of this article.

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 [Grant access to Azure blob and queue data with RBAC in the Azure portal](#).

**NOTE**

Keep in mind that RBAC 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 in Azure Data Lake Storage Gen2](#).

#### Authenticate 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.

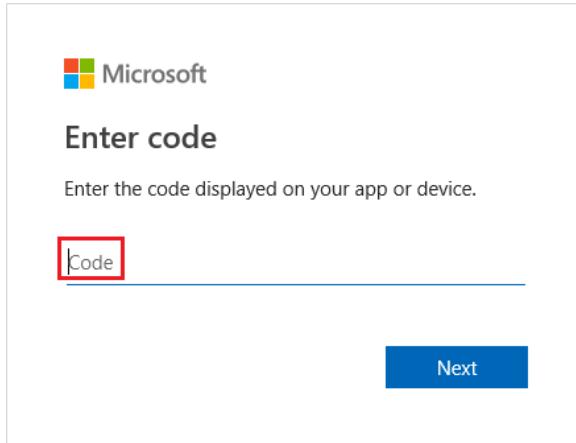
```
azcopy login
```

If you belong to more than one organization, include 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.

#### Authenticate 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 [Authenticate 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](#)

#### 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.

#### 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 login.

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.

#### Authenticate 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 the 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](#).

#### 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
```

#### 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.

#### 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 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=MGCXiyezbttkr3ewJh2AR8KrgbSy1DGM9ovN734bQF4%3D" --recursive=true
```

To learn more about SAS tokens and how to obtain one, see [Using shared access signatures \(SAS\)](#).

## Transfer files

After you've authenticated your identity or obtained a SAS token, you can begin transferring files.

To find example commands, see any of these articles.

- [Transfer data with AzCopy and blob storage](#)
- [Transfer data with AzCopy and file storage](#)
- [Transfer data with AzCopy and Amazon S3 buckets](#)
- [Transfer data with AzCopy and Azure Stack storage](#)

## Use AzCopy 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 (un-changing) 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 -v https://aka.ms/downloadazcopy-v10-linux</pre>
Windows	<pre>(curl https://aka.ms/downloadazcopy-v10-windows -MaximumRedirection 0 -ErrorAction silentlycontinue).RawContent</pre>

#### 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	<pre>wget -O azcopy_v10.tar.gz https://aka.ms/downloadazcopy-v10-linux &amp;&amp; tar -xf azcopy_v10.tar.gz --strip-components=1</pre>

OPERATING SYSTEM	COMMAND
Windows	<pre>Invoke-WebRequest https://azcopyvnext.azureedge.net/release20190517/azcopy_windows_amd -OutFile azcopyv10.zip &lt;&lt;Unzip here&gt;&gt;</pre>

### 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 AzCopy 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.

## Use the previous version of AzCopy

If you need to use the previous version of AzCopy, see either of the following links:

- [AzCopy on Windows \(v8\)](#)
- [AzCopy on Linux \(v7\)](#)

## Configure, optimize, and troubleshoot AzCopy

See [Configure, optimize, and troubleshoot AzCopy](#)

## Next steps

If you have questions, issues, or general feedback, submit them [on GitHub](#) page.

# Transfer data with AzCopy and Blob storage

4/10/2020 • 10 minutes to read • [Edit Online](#)

AzCopy is a command-line utility that you can use to copy data to, from, or between storage accounts. This article contains example commands that work with Blob storage.

## TIP

The examples in this article 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 ('').

## 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 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://<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. The examples in this section create a container named `mycontainer`.

Syntax	<code>azcopy make 'https://&lt;storage-account-name&gt;.&lt;blob or dfs&gt;.core.windows.net/&lt;container-name&gt;'</code>
Example	<code>azcopy make 'https://mystorageaccount.blob.core.windows.net/mycontainer'</code>
Example (hierarchical namespace)	<code>azcopy make 'https://mystorageaccount.dfs.core.windows.net/mycontainer'</code>

For detailed reference docs, see [azcopy make](#).

## Upload files

You can use the [azcopy copy](#) command to upload files and directories from your local computer.

This section contains the following examples:

- Upload a file
- Upload a directory
- Upload the contents of a directory
- Upload specific files

**TIP**

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]
Automatically decompress files.	--decompress=[gzip deflate]

For a complete list, see [options](#).

## Upload a file

Syntax	<code>azcopy copy '&lt;local-file-path&gt;' 'https://&lt;storage-account-name&gt;.&lt;blob or dfs&gt;.core.windows.net/&lt;container-name&gt;/&lt;blob-name&gt;'</code>
Example	<code>azcopy copy 'C:\myDirectory\myTextFile.txt' 'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFi</code>
Example (hierarchical namespace)	<code>azcopy copy 'C:\myDirectory\myTextFile.txt' 'https://mystorageaccount.dfs.core.windows.net/mycontainer/myTextFil</code>

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

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.

Syntax	<code>azcopy copy '&lt;local-directory-path&gt;' 'https://&lt;storage-account-name&gt;.&lt;blob or dfs&gt;.core.windows.net/&lt;container-name&gt;' --recursive</code>
Example	<code>azcopy copy 'C:\myDirectory' 'https://mystorageaccount.blob.core.windows.net/mycontainer' --recursive</code>
Example (hierarchical namespace)	<code>azcopy copy 'C:\myDirectory' 'https://mystorageaccount.dfs.core.windows.net/mycontainer' --recursive</code>

To copy to a directory within the container, just specify the name of that directory in your command string.

Example	<code>azcopy copy 'C:\myDirectory' 'https://mystorageaccount.blob.core.windows.net/mycontainer/myBlobDi</code>
Example (hierarchical namespace)	<code>azcopy copy 'C:\myDirectory' 'https://mystorageaccount.dfs.core.windows.net/mycontainer/myBlobDir</code>

If you specify the name of a directory that does not exist in the container, AzCopy creates a new directory by that name.

## Upload the contents of a directory

You can upload the contents of a directory without copying the containing directory itself by using the wildcard symbol (\*).

Syntax	<code>azcopy copy '&lt;local-directory-path&gt;\*' 'https://&lt;storage-account-name&gt;.&lt;blob or dfs&gt;.core.windows.net/&lt;container-name&gt;/&lt;directory-path&gt;'</code>
--------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

## Example

```
azcopy copy 'C:\myDirectory*' 'https://mystorageaccount.blob.core.windows.net/mycontainer/myBlobDir'
```

## Example (hierarchical namespace)

```
azcopy copy 'C:\myDirectory*' 'https://mystorageaccount.dfs.core.windows.net/mycontainer/myBlobDir'
```

### NOTE

Append the `--recursive` flag to upload files in all sub-directories.

## Upload specific files

You can specify complete file names, or use partial names with wildcard characters (\*).

### 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. You need to 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.

## Download files

You can use the [azcopy copy](#) command to download blobs, directories, and containers to your local computer.

This section contains the following examples:

- Download a file
- Download a directory
- Download the contents of a directory

- Download specific files

#### TIP

You can tweak your download operation by using optional flags. Here's a few examples.

SCENARIO	FLAG
Automatically decompress files.	--decompress=[gzip deflate]
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](#).

#### 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 file

Syntax	<code>azcopy copy 'https://&lt;storage-account-name&gt;.&lt;blob or dfs&gt;.core.windows.net/&lt;container-name&gt;/&lt;blob-path&gt;' '&lt;local-file-path&gt;'</code>
Example	<code>azcopy copy 'https://mystorageaccount.blob.core.windows.net/mycontainer/myTextFile.txt' 'C:\myDirectory\myTextFile.txt'</code>
Example (hierarchical namespace)	<code>azcopy copy 'https://mystorageaccount.dfs.core.windows.net/mycontainer/myTextFile.txt' 'C:\myDirectory\myTextFile.txt'</code>

## Download a directory

Syntax	<code>azcopy copy 'https://&lt;storage-account-name&gt;.&lt;blob or dfs&gt;.core.windows.net/&lt;container-name&gt;/&lt;directory-path&gt;' '&lt;local-directory-path&gt;' --recursive</code>
Example	<code>azcopy copy 'https://mystorageaccount.blob.core.windows.net/mycontainer/myBlobDir' 'C:\myDirectory' --recursive</code>
Example (hierarchical namespace)	<code>azcopy copy 'https://mystorageaccount.dfs.core.windows.net/mycontainer/myBlobDir' 'C:\myDirectory' --recursive</code>

This example results in a directory named `C:\myDirectory\myBlobDirectory` that contains all of the downloaded files.

## Download the contents of a directory

You can download the contents of a directory without copying the containing directory itself by using the wildcard symbol (\*).

#### NOTE

Currently, this scenario is supported only for accounts that don't have a hierarchical namespace.

Syntax	<code>azcopy copy 'https://&lt;storage-account-name&gt;.blob.core.windows.net/&lt;container-name&gt;/*' '&lt;local-directory-path&gt;/'</code>
--------	------------------------------------------------------------------------------------------------------------------------------------------------

## Example

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

## NOTE

Append the `--recursive` flag to download files in all sub-directories.

## Download specific files

You can specify complete file names, or use partial names with wildcard characters (\*).

### 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 '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/FileDire
'C:\myDirectory' --include-path 'photos;documents\myFile.txt' --recu
```

## Example (hierarchical namespace)

```
azcopy copy
'https://mystorageaccount.dfs.core.windows.net/mycontainer/FileDirec
'C:\myDirectory' --include-path 'photos;documents\myFile.txt'--recur
```

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. You need to include the `--recursive` option to transfer all files in the

`https://mystorageaccount.blob.core.windows.net/mycontainer/FileDirectory/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 '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/FileDire
'C:\myDirectory' --include-pattern 'myFile*.txt;*.pdf*'
```

## Example (hierarchical namespace)

```
azcopy copy
'https://mystorageaccount.dfs.core.windows.net/mycontainer/FileDirec
'C:\myDirectory' --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.

## Copy blobs between storage accounts

You can use AzCopy to copy blobs to other storage accounts. The copy operation is synchronous so when the command returns, that indicates that all files have been copied.

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. You can increase the throughput of these operations by setting the value of the `AZCOPY_CONCURRENCY_VALUE` environment variable. To learn more, see [Optimize throughput](#).

**NOTE**

This scenario has the following limitations in the current release.

- You have to append a SAS token to each source URL. If you provide authorization credentials by using Azure Active Directory (AD), you can omit the SAS token only from the destination URL.
- Premium block blob storage accounts don't support access tiers. 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`).

This section contains the following examples:

- Copy a blob to another storage account
- Copy a directory to another storage account
- Copy a container to another storage account
- Copy all containers, directories, and files to another storage account

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.

**TIP**

You can tweak your copy operation by using optional flags. Here's a few examples.

SCENARIO	FLAG
Copy files as Append Blobs or Page Blobs.	<code>--blob-type=[BlockBlob PageBlob AppendBlob]</code>
Copy to a specific access tier (such as the archive tier).	<code>--block-blob-tier=[None Hot Cool Archive]</code>
Automatically decompress files.	<code>--decompress=[gzip deflate]</code>
For a complete list, see <a href="#">options</a> .	

### Copy a blob to another storage account

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

Syntax	<pre>azcopy copy 'https://&lt;source-storage-account-name&gt;.blob.core.windows.net/&lt;container-name&gt;/&lt;blob-path&gt;&lt;SAS-token&gt;' 'https://&lt;destination-storage-account-name&gt;.blob.core.windows.net/&lt;container-name&gt;/&lt;blob-path&gt;'</pre>
Example	<pre>azcopy copy 'https://mysourceaccount.blob.core.windows.net/mycontainer/myTextFile?sv=2018-03-28&amp;ss=bfqt&amp;srt=sco&amp;sp=rw&amp;st=2019-07-04T05:30:08Z&amp;st=2019-07-04T05:30:08Z&amp;spr=https&amp;sig=CAFhgnc9gdGktvB=ska7bAiqIdM845yiFwdMH48 'https://mydestinationaccount.blob.core.windows.net/mycontainer/myTextFile?sv=2018-03-28&amp;ss=bfqt&amp;srt=sco&amp;sp=rw&amp;st=2019-07-04T05:30:08Z&amp;st=2019-07-04T05:30:08Z&amp;spr=https&amp;sig=CAFhgnc9gdGktvB=ska7bAiqIdM845yiFwdMH48</pre>
Example (hierarchical namespace)	<pre>azcopy copy 'https://mysourceaccount.blob.core.windows.net/mycontainer/myTextFile?sv=2018-03-28&amp;ss=bfqt&amp;srt=sco&amp;sp=rw&amp;st=2019-07-04T05:30:08Z&amp;st=2019-07-04T05:30:08Z&amp;spr=https&amp;sig=CAFhgnc9gdGktvB=ska7bAiqIdM845yiFwdMH48 'https://mydestinationaccount.blob.core.windows.net/mycontainer/myTextFile?sv=2018-03-28&amp;ss=bfqt&amp;srt=sco&amp;sp=rw&amp;st=2019-07-04T05:30:08Z&amp;st=2019-07-04T05:30:08Z&amp;spr=https&amp;sig=CAFhgnc9gdGktvB=ska7bAiqIdM845yiFwdMH48</pre>

### Copy a directory to another storage account

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

Syntax	<pre>azcopy copy 'https://&lt;source-storage-account-name&gt;.blob.core.windows.net/&lt;container-name&gt;/&lt;directory-path&gt;&lt;SAS-token&gt;' 'https://&lt;destination-storage-account-name&gt;.blob.core.windows.net/&lt;container-name&gt;' --recursive</pre>
Example	<pre>azcopy copy 'https://mysourceaccount.blob.core.windows.net/mycontainer/myBlobDir?sv=2018-03-28&amp;ss=bfqt&amp;srt=sco&amp;sp=rw&amp;st=2019-07-04T05:30:08Z&amp;st=2019-07-04T05:30:08Z&amp;spr=https&amp;sig=CAFhgnc9gdGktvB=ska7bAiqIdM845yiFwdMH48 'https://mydestinationaccount.blob.core.windows.net/mycontainer' --recursive</pre>

### Example (hierarchical namespace)

```
azcopy copy
'https://mysourceaccount.blob.core.windows.net/mycontainer/myBlobDir
sv=2018-03-28&ss=bfqt&srt=sco&sp=rwdracup&se=2019-07-04T05:30:08Z&st
07-
03T21:30:08Z&spr=https&sig=CAFhgn9gdGktvB=ska7bAiqIdM845yiyFwdMH48
'https://mydestinationaccount.blob.core.windows.net/mycontainer' --r
```

## Copy a container to another storage account

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

### Syntax

```
azcopy copy '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 copy 'https://mysourceaccount.blob.core.windows.net/mycontain
sv=2018-03-28&ss=bfqt&srt=sco&sp=rwdracup&se=2019-07-04T05:30:08Z&st
07-
03T21:30:08Z&spr=https&sig=CAFhgn9gdGktvB=ska7bAiqIdM845yiyFwdMH48
'https://mydestinationaccount.blob.core.windows.net/mycontainer' --r
```

### Example (hierarchical namespace)

```
azcopy copy 'https://mysourceaccount.blob.core.windows.net/mycontain
sv=2018-03-28&ss=bfqt&srt=sco&sp=rwdracup&se=2019-07-04T05:30:08Z&st
07-
03T21:30:08Z&spr=https&sig=CAFhgn9gdGktvB=ska7bAiqIdM845yiyFwdMH48
'https://mydestinationaccount.blob.core.windows.net/mycontainer' --r
```

## Copy all containers, directories, and blobs to another storage account

Use the same URL syntax (`blob.core.windows.net`) for accounts that have a hierarchical namespace.

### Syntax

```
azcopy copy 'https://<source-storage-account-
name>.blob.core.windows.net/<SAS-token>'
'https://<destination-storage-account-
name>.blob.core.windows.net/' --recursive
```

### Example

```
azcopy copy 'https://mysourceaccount.blob.core.windows.net/?sv=2018-
28&ss=bfqt&srt=sco&sp=rwdracup&se=2019-07-04T05:30:08Z&st=2019-07-
03T21:30:08Z&spr=https&sig=CAFhgn9gdGktvB=ska7bAiqIdM845yiyFwdMH48
'https://mydestinationaccount.blob.core.windows.net' --recursive
```

### Example (hierarchical namespace)

```
azcopy copy 'https://mysourceaccount.blob.core.windows.net/?sv=2018-
28&ss=bfqt&srt=sco&sp=rwdracup&se=2019-07-04T05:30:08Z&st=2019-07-
03T21:30:08Z&spr=https&sig=CAFhgn9gdGktvB=ska7bAiqIdM845yiyFwdMH48
'https://mydestinationaccount.blob.core.windows.net' --recursive
```

## Synchronize files

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).

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`.

### NOTE

To prevent accidental deletions, make sure to enable the `soft delete` feature before you use the `--delete-destination=prompt|true` flag.

**TIP**

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]
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, see <a href="#">options</a> .	

### 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.

Syntax	<pre>azcopy sync '&lt;local-directory-path&gt;' 'https://&lt;storage-account-name&gt;.blob.core.windows.net/&lt;container-name&gt;' --recursive</pre>
Example	<pre>azcopy sync 'C:\myDirectory' 'https://mystorageaccount.blob.core.windows.net/mycontainer' --recursive</pre>

### 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.

Syntax	<pre>azcopy sync 'https://&lt;storage-account-name&gt;.blob.core.windows.net/&lt;container-name&gt;' 'C:\myDirectory' --recursive</pre>
Example	<pre>azcopy sync 'https://mystorageaccount.blob.core.windows.net/mycontainer' 'C:\myDirectory' --recursive</pre>

### 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.

Syntax	<pre>azcopy sync 'https://&lt;source-storage-account-name&gt;.blob.core.windows.net/&lt;container-name&gt;' 'https://&lt;destination-storage-account-name&gt;.blob.core.windows.net/&lt;container-name&gt;' --recursive</pre>
Example	<pre>azcopy sync 'https://mysourceaccount.blob.core.windows.net/mycontainer' 'https://mydestinationaccount.blob.core.windows.net/mycontainer' --recursive</pre>

### Update a directory with changes to a directory in another file share

The first directory that appears in this command is the source. The second one is the destination.

Syntax	<pre>azcopy sync 'https://&lt;source-storage-account-name&gt;.blob.core.windows.net/&lt;container-name&gt;/&lt;directory-name&gt;' 'https://&lt;destination-storage-account-name&gt;.blob.core.windows.net/&lt;container-name&gt;/&lt;directory-name&gt;' --recursive</pre>
Example	<pre>azcopy sync 'https://mysourceaccount.blob.core.windows.net/&lt;container-name&gt;/myDirectory' 'https://mydestinationaccount.blob.core.windows.net/mycontainer/myDi--recursive</pre>

## Next steps

Find more examples in any of these articles:

- [Get started with AzCopy](#)
- [Tutorial: Migrate on-premises data to cloud storage by using AzCopy](#)
- [Transfer data with AzCopy and file storage](#)
- [Transfer data with AzCopy and Amazon S3 buckets](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

# Configure, optimize, and troubleshoot AzCopy

4/10/2020 • 7 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 perform advanced configuration tasks and helps you to troubleshoot issues that can arise as you use AzCopy.

## NOTE

If you're looking for content to help you get started with AzCopy, see any of the following 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](#)

## 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: <code>set https_proxy=&lt;proxy IP&gt;:&lt;proxy port&gt;</code> In PowerShell use: <code>\$env:https_proxy="&lt;proxy IP&gt;:&lt;proxy port&gt;"</code>
Linux	<code>export https_proxy=&lt;proxy IP&gt;:&lt;proxy port&gt;</code>
MacOS	<code>export https_proxy=&lt;proxy IP&gt;:&lt;proxy port&gt;</code>

Currently, AzCopy doesn't support proxies that require authentication with NTLM or Kerberos.

## Optimize performance

You can benchmark performance, and then use commands and environment variables to find an optimal tradeoff between performance and resource consumption.

This section helps you perform these optimization tasks:

- Run benchmark tests
- Optimize throughput
- Optimize memory use
- Optimize file synchronization

### Run benchmark tests

You can run a performance benchmark test on specific blob containers to view general performance statistics and to identify performance bottlenecks.

Use the following command to run a performance benchmark test.

Syntax	<code>azcopy bench 'https://&lt;storage-account-name&gt;.blob.core.windows.net/&lt;container-name&gt;'</code>
Example	<code>azcopy bench 'https://mystorageaccount.blob.core.windows.net/mycontainer/myBlobDir?sv=2018-03-28&amp;ss=bjqt&amp;srs=sco&amp;sp=rjk1hjup&amp;se=2019-05-10T04:37:48Z&amp;st=2019T20:37:48Z&amp;spr=https&amp;sig=%2F5OVELfsKDqRry4bk3qz1vAQFwY5DDzp2%2B%2F3E'</code>

**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.

For detailed reference docs, see [azcopy bench](#).

To view detailed help guidance for this command, type `azcopy bench -h` and then press the ENTER key.

### Optimize throughput

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
```

Throughput can decrease when transferring small files. 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 `3000`, 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.

### Optimize memory use

Set the `AZCOPY_BUFFER_GB` environment variable to specify the maximum amount of your system memory you want AzCopy to use 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>

### 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`.

## Troubleshoot issues

AzCopy creates log and plan files for every job. You can use the 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 if you want.

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
```

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>
azcopy jobs resume <job-id> --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 the plan and log files

By default, plan and log files are located in the `%USERPROFILE%\azcopy` directory on Windows, or in the `$HOME$\azcopy` directory on Mac and Linux. You can change this location.

### Change the location of plan files

Use any of these commands.

OPERATING SYSTEM	COMMAND
Windows	<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	<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: `NONE`, `DEBUG`, `INFO`, `WARNING`, `ERROR`, `PANIC`, and `FATAL`.

## 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.

# Copy and transform data in Azure Blob storage by using Azure Data Factory

4/15/2020 • 26 minutes to read • [Edit Online](#)

APPLIES TO: Azure Data Factory Azure Synapse Analytics (Preview)

This article outlines how to use Copy Activity in Azure Data Factory to copy data from and to Azure Blob storage, and use Data Flow to transform data in Azure Blob storage. To learn about Azure Data Factory, read the [introductory article](#).

## TIP

For data lake or data warehouse migration scenario, learn more from [Use Azure Data Factory to migrate data from your data lake or data warehouse to Azure](#).

## Supported capabilities

This Azure Blob connector is supported for the following activities:

- [Copy activity](#) with [supported source/sink matrix](#)
- [Mapping data flow](#)
- [Lookup activity](#)
- [GetMetadata activity](#)
- [Delete activity](#)

For 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 account key, service shared access signature, service principal or managed identities for Azure resources 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](#).
- [Preserve file metadata during copy](#).

## IMPORTANT

If you enable the **Allow trusted Microsoft services to access this storage account** option on Azure Storage firewall settings and want to use Azure integration runtime to connect to your Blob Storage, you must use [managed identity authentication](#).

## Get started

You can use one of the following tools or SDKs to use the copy activity with a pipeline. Select a link for step-by-step instructions:

- [Copy data tool](#)
- [Azure portal](#)
- [.NET SDK](#)

- [Python SDK](#)
- [Azure PowerShell](#)
- [REST API](#)
- [Azure Resource Manager template](#)

The following sections provide details about properties that are used to define Data Factory entities specific to Blob storage.

## Linked service properties

Azure Blob connector support the following authentication types, refer to the corresponding section on details:

- [Account key authentication](#)
- [Shared access signature authentication](#)
- [Service principal authentication](#)
- [Managed identities for Azure resources authentication](#)

### NOTE

When using PolyBase to load data into SQL Data Warehouse, if your source or staging Blob storage is configured with Virtual Network endpoint, you must use managed identity authentication as required by PolyBase, and use Self-hosted Integration Runtime with version 3.18 or above. See the [managed identity authentication](#) section with more configuration prerequisites.

### NOTE

HDIInsights and Azure Machine Learning activities only support Azure Blob storage account key authentication.

### 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>AzureBlobStorage</b> (suggested) or <b>AzureStorage</b> (see notes below).	Yes
connectionString	Specify the information needed to connect to Storage for the connectionString property. You can also put account key in Azure Key Vault and pull the <code>accountKey</code> configuration out of the connection string. Refer to the following samples and <a href="#">Store credentials in Azure Key Vault</a> article with more details.	Yes
connectVia	The <a href="#">integration runtime</a> to be used to connect to the data store. You can use Azure Integration Runtime or Self-hosted Integration Runtime (if your data store is in a private network). If not specified, it uses the default Azure Integration Runtime.	No

**NOTE**

Secondary Blob Service Endpoint is not supported when using account key authentication. You can use other authentication types.

**NOTE**

If you were using "AzureStorage" type linked service, it is still supported as-is, while you are suggested to use this 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 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**

- Data Factory 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 \(SAS\)](#).
- In later dataset configuration, the folder path is the absolute path starting from container level. You need to configure one aligned with the path in your SAS URI.

To use shared access signature authentication, the following properties are supported:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property must be set to <b>AzureBlobStorage</b> (suggested) or <b>AzureStorage</b> (see notes below).	Yes
sasUri	Specify the shared access signature URI to the Storage resources such as blob/container. Mark this field as a SecureString to store it securely in Data Factory. You can also put SAS token in Azure Key Vault to leverage auto rotation and remove the token portion. Refer to the following samples and <a href="#">Store credentials in Azure Key Vault</a> article with more details.	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 located in a private network). If not specified, it uses the default Azure Integration Runtime.	No

**NOTE**

If you were using "AzureStorage" type linked service, it is still supported as-is, while you are suggested to use this 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://<container>.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 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://<container>.blob.core.windows.net/>"

 },
 "sasToken": {
 "type": "AzureKeyVaultSecret",
 "store": {
 "referenceName": "<Azure Key Vault linked service name>",
 "type": "LinkedServiceReference"
 },
 "secretName": "<secretName>"
 }
 },
 "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 in your data factory.
- 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/blob based on the need. A shared access signature URI to a blob allows Data Factory to access that particular blob. A shared access signature URI to a Blob storage container allows Data Factory 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 Azure Storage service principal authentication in general, refer to [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 the following 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. Refer to [Manage access rights to Azure Storage data with RBAC](#) with more details on the roles.

- **As source**, in Access control (IAM), grant at least **Storage Blob Data Reader** role.
- **As sink**, in Access control (IAM), grant at least **Storage Blob Data Contributor** role.

These properties are supported for an Azure Blob storage linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type 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
servicePrincipalId	Specify the application's client ID.	Yes
servicePrincipalKey	Specify the application's key. Mark this field as a <b>SecureString</b> to store it securely in Data Factory, 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 the mouse in the top-right corner of the Azure portal.	Yes
connectVia	The <a href="#">integration runtime</a> to be used to connect to the data store. You can use Azure Integration Runtime or Self-hosted Integration Runtime (if your data store is in a private network). If not specified, it uses the default Azure Integration Runtime.	No

### NOTE

Service principal authentication is only supported by "AzureBlobStorage" type linked service but not previous "AzureStorage" type linked service.

**Example:**

```
{
 "name": "AzureBlobStorageLinkedService",
 "properties": {
 "type": "AzureBlobStorage",
 "typeProperties": {
 "serviceEndpoint": "https://<accountName>.blob.core.windows.net/",
 "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"
 }
 }
}
```

## Managed identities for Azure resources authentication

A data factory can be associated with a [managed identity for Azure resources](#), which represents this specific data factory. You can directly use this managed identity for Blob storage authentication similar to using your own service principal. It allows this designated factory to access and copy data from/to your Blob storage.

Refer to [Authenticate access to Azure Storage using Azure Active Directory](#) for Azure Storage authentication in general. To use managed identities for Azure resources authentication, follow these steps:

1. [Retrieve data factory managed identity information](#) by copying the value of **managed identity object ID** generated along with your factory.
2. Grant the managed identity proper permission in Azure Blob storage. Refer to [Manage access rights to Azure Storage data with RBAC](#) with more details on the roles.
  - As **source**, in Access control (IAM), grant at least **Storage Blob Data Reader** role.
  - As **sink**, in Access control (IAM), grant at least **Storage Blob Data Contributor** role.

### IMPORTANT

If you use PolyBase to load data from Blob (as source or as staging) into SQL Data Warehouse, when using managed identity authentication for Blob, make sure you also follow steps 1 and 2 in [this guidance](#) to 1) register your SQL Database server with Azure Active Directory (Azure AD) and 2) assign the Storage Blob Data Contributor role to your SQL Database server; the rest are handled by Data Factory. If your Blob storage is configured with an Azure Virtual Network endpoint, to use PolyBase to load data from it, you must use managed identity authentication as required by PolyBase.

These properties are supported for an Azure Blob storage linked service:

PROPERTY	DESCRIPTION	REQUIRED
type	The type 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
connectVia	The <a href="#">integration runtime</a> to be used to connect to the data store. You can use Azure Integration Runtime or Self-hosted Integration Runtime (if your data store is in a private network). If not specified, it uses the default Azure Integration Runtime.	No

#### NOTE

Managed identities for Azure resources authentication is only supported by "AzureBlobStorage" type linked service but not previous "AzureStorage" type linked service.

#### Example:

```
{
 "name": "AzureBlobStorageLinkedService",
 "properties": {
 "type": "AzureBlobStorage",
 "typeProperties": {
 "serviceEndpoint": "https://<accountName>.blob.core.windows.net/"
 },
 "connectVia": {
 "referenceName": "<name of Integration Runtime>",
 "type": "IntegrationRuntimeReference"
 }
 }
}
```

## Dataset properties

For a full list of sections and properties available for defining datasets, see the [Datasets](#) article.

Azure Data Factory support the following file formats. Refer to each article on format-based settings.

- [Avro format](#)
- [Binary format](#)
- [Delimited text format](#)
- [JSON format](#)
- [ORC format](#)
- [Parquet format](#)

The following properties are supported for Azure Blob under `location` settings in format-based dataset:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the location in dataset must be set to <a href="#">AzureBlobStorageLocation</a> .	Yes
container	The blob container.	Yes

PROPERTY	DESCRIPTION	REQUIRED
folderPath	The path to folder under the given container. If you want to use wildcard to filter folder, skip this setting and specify in activity source settings.	No
fileName	The file name under the given container + folderPath. If you want to use wildcard to filter files, skip this setting and specify 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 supported by the Blob storage source and sink.

### Blob storage as a source type

Azure Data Factory support the following file formats. Refer to each article on format-based settings.

- [Avro format](#)
- [Binary format](#)
- [Delimited text format](#)
- [JSON format](#)
- [ORC format](#)
- [Parquet format](#)

The following properties are supported for Azure Blob under `storeSettings` settings in format-based copy source:

PROPERTY	DESCRIPTION	REQUIRED
----------	-------------	----------

PROPERTY	DESCRIPTION	REQUIRED
type	The type property under <code>storeSettings</code> must be set to <code>AzureBlobStorageReadSettings</code> .	Yes
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 <code>true</code> (default) and <code>false</code> .	No
prefix	Prefix for the blob name under the given container configured in dataset to filter source blobs. Blobs whose name starts with this prefix are selected. Applies only when <code>wildcardFolderPath</code> and <code>wildcardFileName</code> properties are not specified.	No
wildcardFolderPath	The folder path with wildcard characters under the given container 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
wildcardFileName	The file name with wildcard characters under the given container + <code>folderPath/wildcardFolderPath</code> 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 folder name has wildcard or this escape char inside. See more examples in <a href="#">Folder and file filter examples</a> .	Yes if <code>fileName</code> is not specified in dataset

PROPERTY	DESCRIPTION	REQUIRED
modifiedDatetimeStart	<p>Files filter based on the attribute: Last Modified. The files will be selected if their last modified time are within the time range between <code>modifiedDatetimeStart</code> and <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 mean 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. 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>	No
modifiedDatetimeEnd	Same as above.	No
maxConcurrentConnections	The number of the connections to connect to storage store concurrently. Specify only when you want to limit the concurrent connection to the data store.	No

#### NOTE

For Parquet/delimited text format, **BlobSource** type copy activity source mentioned in next section is still supported as-is for backward compatibility. You are suggested to use this new model going forward, and the ADF 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>"
 }
 }
 }
]

```

## Blob storage as a sink type

Azure Data Factory support the following file formats. Refer to each article on format-based settings.

- [Avro format](#)
- [Binary format](#)
- [Delimited text format](#)
- [JSON format](#)
- [ORC format](#)
- [Parquet format](#)

The following properties are supported for Azure Blob under `storeSettings` settings in format-based copy sink:

PROPERTY	DESCRIPTION	REQUIRED
type	The type property under <code>storeSettings</code> must be set to <code>AzureBlobStorageWriteSettings</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 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
blockSizeInMB	<p>Specify the block size in MB used to write data to block blobs. Learn more <a href="#">about Block Blobs</a>.</p> <p>Allowed value is <b>between 4 and 100 MB</b>.</p> <p>By default, ADF automatically determine the block size based on your source store type and data. For non-binary copy into Blob, the default block size is 100 MB so as to fit in at most 4.95 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 blockSizeInMB*50000 is big enough to store the data, otherwise copy activity run will fail.</p>	No
maxConcurrentConnections	The number of the connections to connect to storage store concurrently. Specify only when you want to limit the concurrent connection to the data store.	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"
 }
 }
 }
 }
]

```

## 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 <b>File3.csv</b> <b>File4.json</b> <b>File5.csv</b> AnotherFolderB <b>File6.csv</b>
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>

FOLDERPATH	FILENAME	RECURSIVE	SOURCE FOLDER STRUCTURE AND FILTER RESULT (FILES IN BOLD ARE RETRIEVED)
container/Folder*	*.csv	false	container FolderA <b>File1.csv</b> File2.json Subfolder1 File3.csv <b>File4.json</b> File5.csv AnotherFolderB File6.csv
container/Folder*	*.csv	true	container FolderA <b>File1.csv</b> File2.json Subfolder1 <b>File3.csv</b> File4.json <b>File5.csv</b> AnotherFolderB File6.csv

### 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 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

Recursive	CopyBehavior	Source Folder Structure	Resulting Target
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 folder Folder1 is created with the following structure:  Folder1 File1 File2  Subfolder1 with File3, File4, and File5 is not picked up.
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.

## 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](#).

## Mapping data flow properties

When transforming data in mapping data flow, you can read and write files from Azure Blob Storage in JSON, Avro, Delimited Text, or Parquet format. For more information, see [source transformation](#) and [sink transformation](#)

in the mapping data flow feature.

## Source transformation

In the source transformation, you can read from a container, folder or individual file in Azure Blob Storage. 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. The configuration includes:

- Wildcard paths:** mycontainer / partdata/\*\*/\*.csv
- Partition root path:** partdata
- List of files:** (checkbox)
- Multiline rows:** (checkbox)
- Column to store file name:** fileName
- After completion:** No action (selected)
- Start time (UTC):** 02/01/2020 00:00:00
- End time (UTC):** 02/02/2020 00:00:00
- Filter by last modified:** (checkbox)

**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 i + -

Partition root path partdata i

List of files i

Multiline rows i

Column to store file name myfile i

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 <span style="color: green;">●</span>	Descript
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 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.

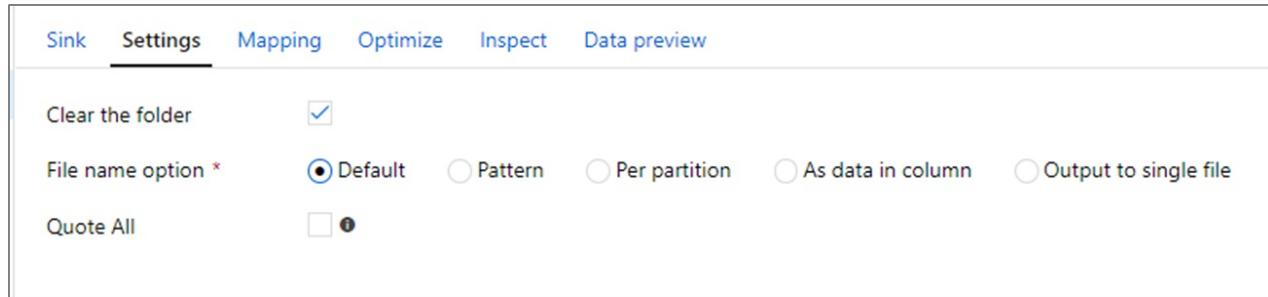
#### NOTE

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 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.

#### Sink properties

In the sink transformation, you can write to either a container or folder in Azure Blob Storage. the **Settings** tab lets you manage how the files get written.



**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

## 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
type	The type property of the dataset must be set to <b>AzureBlob</b> .	Yes
folderPath	<p>Path to the container and folder in the blob storage.</p> <p>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 actual folder name has wildcard or this escape char inside.</p> <p>Examples: myblobcontainer/myblobfolder/, see more examples in <a href="#">Folder and file filter examples</a>.</p>	Yes for Copy/Lookup activity, No for GetMetadata activity
fileName	<p><b>Name or wildcard filter</b> for the blob(s) under the specified "FolderPath". If you don't specify a value for this property, the dataset points to all blobs in the folder.</p> <p>For 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 actual file name has wildcard or this escape char 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 blob name with the following pattern: <code>"Data.[activity run ID GUID].[GUID if FlattenHierarchy].[format if configured].[compression if configured]"</code>, e.g. <code>"Data.0a405f8a-93ff-4c6f-b3be-f69616f1df7a.txt.gz"</code>; if you copy from tabular source using table name instead of query, the name pattern is <code>"[table name].[format].[compression if configured]"</code>, e.g. <code>"MyTable.csv"</code>.</p>	No

PROPERTY	DESCRIPTION	REQUIRED
modifiedDatetimeStart	<p>Files filter based on the attribute: Last Modified. The files will be selected if their last modified time are within the time range between <code>modifiedDatetimeStart</code> and <code>modifiedDatetimeEnd</code>. The time is applied to UTC time zone in the format of "2018-12-01T05:00:00Z".</p> <p>Be aware the overall performance of data movement will be impacted by enabling this setting when you want to do file filter from huge amounts of files.</p> <p>The properties can be NULL that mean 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. 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>	No
modifiedDatetimeEnd	<p>Files filter based on the attribute: Last Modified. The files will be selected if their last modified time are within the time range between <code>modifiedDatetimeStart</code> and <code>modifiedDatetimeEnd</code>. The time is applied to UTC time zone in the format of "2018-12-01T05:00:00Z".</p> <p>Be aware the overall performance of data movement will be impacted by enabling this setting when you want to do file filter from huge amounts of files.</p> <p>The properties can be NULL that mean 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. 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>	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 <code>type</code> property under <code>format</code> 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 <a href="#">GZip</a>, <a href="#">Deflate</a>, <a href="#">BZip2</a>, and <a href="#">ZipDeflate</a>.</p> <p>Supported levels are <a href="#">Optimal</a> and <a href="#">Fastest</a>.</p>	No

#### TIP

To copy all blobs under a folder, specify `FolderPath` only.

To copy a single blob with a given name, specify `FolderPath` with folder part and `fileName` with file name.

To copy a subset of blobs under a folder, specify `FolderPath` with folder part and `fileName` with 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 copy activity source model

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity source must be set to <b>BlobSource</b> .	Yes
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> .	No
maxConcurrentConnections	The number of the connections to connect to storage store concurrently. Specify only when you want to limit the concurrent connection to the data store.	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 copy activity sink model

PROPERTY	DESCRIPTION	REQUIRED
type	The type property of the copy activity sink must be set to <b>BlobSink</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 (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 number of the connections to connect to storage store concurrently. Specify only when you want to limit the concurrent connection to the data store.	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"
 }
 }
 }
]
```

## Next steps

For a list of data stores supported as sources and sinks by the copy activity in Data Factory, see [Supported data stores](#).

# Transfer data with the Data Movement library

3/10/2020 • 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 .NET Core. 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  $8 *$  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, 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(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.](#)

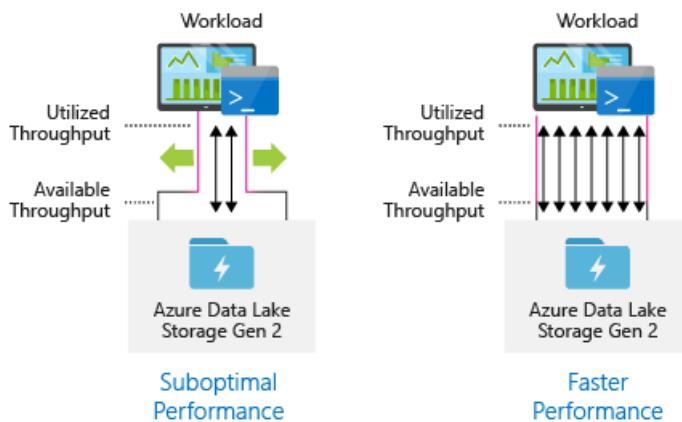
**TIP**

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.

# Optimize Azure Data Lake Storage Gen2 for performance

11/21/2019 • 6 minutes to read • [Edit Online](#)

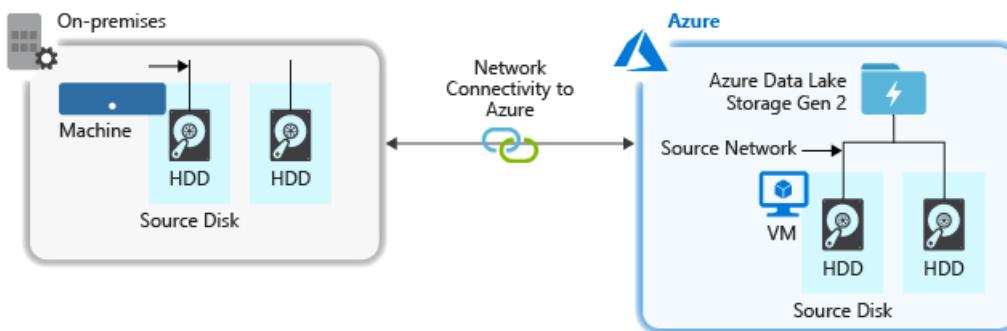
Azure Data Lake Storage Gen2 supports high-throughput for I/O intensive analytics and data movement. In Data Lake Storage Gen2, using all available throughput – the amount of data that can be read or written per second – is important to get the best performance. This is achieved by performing as many reads and writes in parallel as possible.



Data Lake Storage Gen2 can scale to provide the necessary throughput for all analytics scenario. By default, a Data Lake Storage Gen2 account provides automatically enough throughput to meet the needs of a broad category of use cases. For the cases where customers run into the default limit, the Data Lake Storage Gen2 account can be configured to provide more throughput by contacting [Azure Support](#).

## Data ingestion

When ingesting data from a source system to Data Lake Storage Gen2, it is important to consider that the source hardware, source network hardware, and network connectivity to Data Lake Storage Gen2 can be the bottleneck.



It is important to ensure that the data movement is not affected by these factors.

### Source hardware

Whether you are using on-premises machines or VMs in Azure, you should carefully select the appropriate hardware. For Source Disk Hardware, prefer SSDs to HDDs and pick disk hardware with faster spindles. For Source Network Hardware, use the fastest NICs possible. On Azure, we recommend Azure D14 VMs which have the appropriately powerful disk and networking hardware.

### Network connectivity to Data Lake Storage Gen2

The network connectivity between your source data and Data Lake Storage Gen2 can sometimes be the bottleneck.

When your source data is On-Premises, consider using a dedicated link with [Azure ExpressRoute](#). If your source data is in Azure, the performance will be best when the data is in the same Azure region as the Data Lake Storage Gen2 account.

### Configure data ingestion tools for maximum parallelization

Once you have addressed the source hardware and network connectivity bottlenecks above, you are ready to configure your ingestion tools. The following table summarizes the key settings for several popular ingestion tools and provides in-depth performance tuning articles for them. To learn more about which tool to use for your scenario, visit this [article](#).

TOOL	SETTINGS	MORE DETAILS
DistCp	-m (mapper)	<a href="#">Link</a>
Azure Data Factory	parallelCopies	<a href="#">Link</a>
Sqoop	fs.azure.block.size, -m (mapper)	<a href="#">Link</a>

## Structure your data set

When data is stored in Data Lake Storage Gen2, the file size, number of files, and folder structure have an impact on performance. The following section describes best practices in these areas.

### File size

Typically, analytics engines such as HDInsight and Azure Data Lake Analytics have a per-file overhead. 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 (256MB to 100GB in size). Some engines and applications might have trouble efficiently processing files that are greater than 100GB in size.

Sometimes, data pipelines have limited control over the raw data which has lots of small files. It is recommended to have a "cooking" process that generates larger files to use for downstream applications.

### Organizing time series data in folders

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 very structured naming for files and folders. Below is a very 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.

## Optimizing I/O intensive jobs on Hadoop and Spark workloads on HDInsight

Jobs fall into one of the following three categories:

- **CPU intensive.** These jobs have long computation times with minimal I/O times. Examples include machine learning and natural language processing jobs.
- **Memory intensive.** These jobs use lots of memory. Examples include PageRank and real-time analytics jobs.
- **I/O intensive.** These jobs spend most of their time doing I/O. A common example is a copy job which does only read and write operations. Other examples include data preparation jobs that read a lot of data, performs some data transformation, and then writes the data back to the store.

The following guidance is only applicable to I/O intensive jobs.

## General considerations

You can have a job that reads or writes as much as 100MB in a single operation, but a buffer of that size might compromise performance. To optimize performance, try to keep the size of an I/O operation between 4MB and 16MB.

### General considerations for an HDInsight cluster

- **HDInsight versions.** For best performance, use the latest release of HDInsight.
- **Regions.** Place the Data Lake Storage Gen2 account in the same region as the HDInsight cluster.

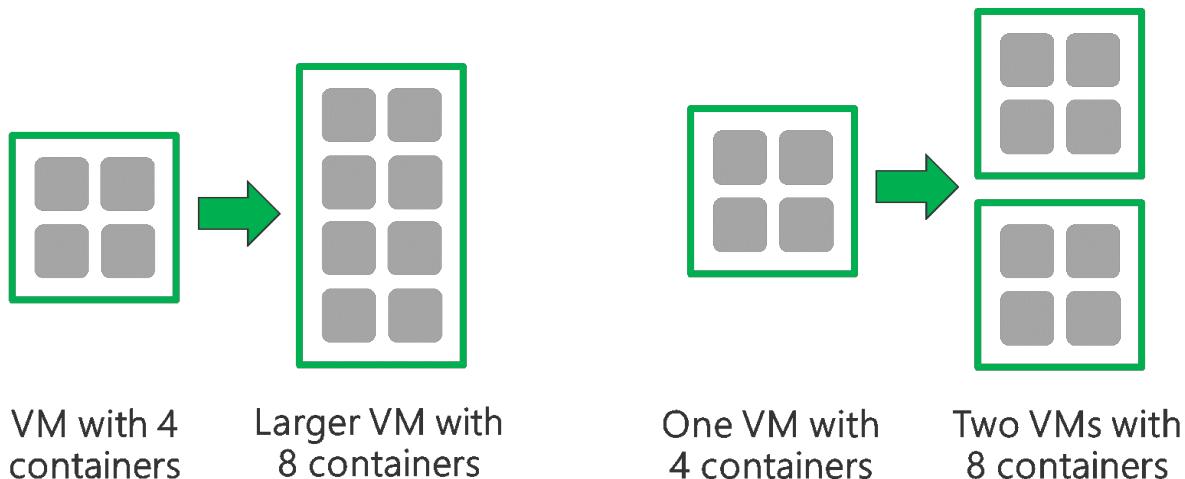
An HDInsight cluster is composed of two head nodes and some worker nodes. Each worker node provides a specific number of cores and memory, which is determined by the VM-type. When running a job, YARN is the resource negotiator that allocates the available memory and cores to create containers. Each container runs the tasks needed to complete the job. Containers run in parallel to process tasks quickly. Therefore, performance is improved by running as many parallel containers as possible.

There are three layers within an HDInsight cluster that can be tuned to increase the number of containers and use all available throughput.

- **Physical layer**
- **YARN layer**
- **Workload layer**

### Physical Layer

**Run cluster with more nodes and/or larger sized VMs.** A larger cluster will enable you to run more YARN containers as shown in the picture below.

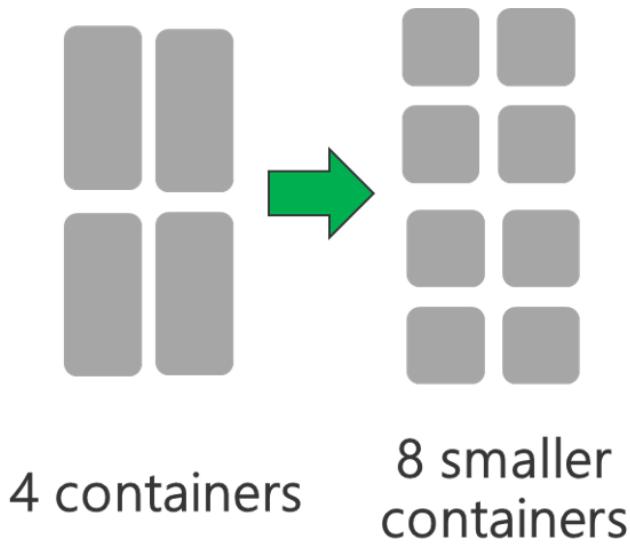


**Use VMs with more network bandwidth.** The amount of network bandwidth can be a bottleneck if there is less network bandwidth than Data Lake Storage Gen2 throughput. Different VMs will have varying network

bandwidth sizes. Choose a VM-type that has the largest possible network bandwidth.

### YARN Layer

**Use smaller YARN containers.** Reduce the size of each YARN container to create more containers with the same amount of resources.

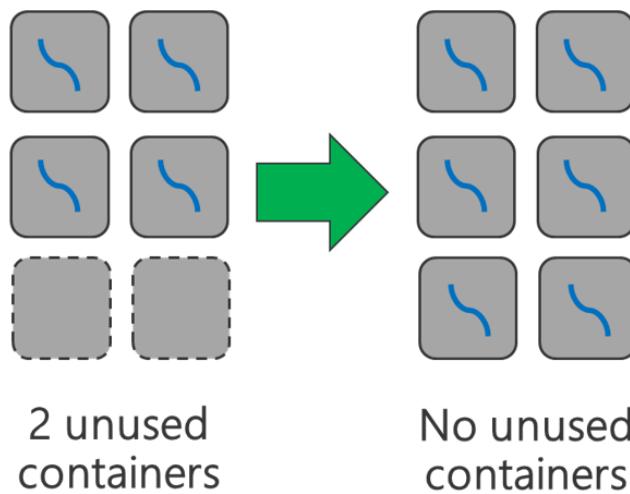


Depending on your workload, there will always be a minimum YARN container size that is needed. If you pick too small a container, your jobs will run into out-of-memory issues. Typically YARN containers should be no smaller than 1GB. It's common to see 3GB YARN containers. For some workloads, you may need larger YARN containers.

**Increase cores per YARN container.** Increase the number of cores allocated to each container to increase the number of parallel tasks that run in each container. This works for applications like Spark which run multiple tasks per container. For applications like Hive which run a single thread in each container, it is better to have more containers rather than more cores per container.

### Workload Layer

**Use all available containers.** Set the number of tasks to be equal or larger than the number of available containers so that all resources are utilized.



**Failed tasks are costly.** If each task has a large amount of data to process, then failure of a task results in an expensive retry. Therefore, it is better to create more tasks, each of which processes a small amount of data.

In addition to the general guidelines above, each application has different parameters available to tune for that

specific application. The table below lists some of the parameters and links to get started with performance tuning for each application.

WORKLOAD	PARAMETER TO SET TASKS
Spark on HDInsight	<ul style="list-style-type: none"><li>• Num-executors</li><li>• Executor-memory</li><li>• Executor-cores</li></ul>
Hive on HDInsight	<ul style="list-style-type: none"><li>• hive.tez.container.size</li></ul>
MapReduce on HDInsight	<ul style="list-style-type: none"><li>• Mapreduce.map.memory</li><li>• Mapreduce.job.maps</li><li>• Mapreduce.reduce.memory</li><li>• Mapreduce.job.reduces</li></ul>
Storm on HDInsight	<ul style="list-style-type: none"><li>• Number of worker processes</li><li>• Number of spout executor instances</li><li>• Number of bolt executor instances</li><li>• Number of spout tasks</li><li>• Number of bolt tasks</li></ul>

## See also

- [Overview of Azure Data Lake Storage Gen2](#)

# Tune performance: Spark, HDInsight & Azure Data Lake Storage Gen2

11/21/2019 • 7 minutes to read • [Edit Online](#)

When tuning performance on Spark, you need to consider the number of apps that will be running on your cluster. By default, you can run 4 apps concurrently on your HDI cluster (Note: the default setting is subject to change). You may decide to use fewer apps so you can override the default settings and use more of the cluster for those apps.

## Prerequisites

- An Azure subscription. See [Get Azure free trial](#).
- An **Azure Data Lake Storage Gen2 account**. For instructions on how to create one, see [Quickstart: Create an Azure Data Lake Storage Gen2 storage account](#).
- Azure **HDInsight cluster** with access to a Data Lake Storage Gen2 account. See [Use Azure Data Lake Storage Gen2 with Azure HDInsight clusters](#). Make sure you enable Remote Desktop for the cluster.
- Running **Spark cluster on Data Lake Storage Gen2**. For more information, see [Use HDInsight Spark cluster to analyze data in Data Lake Storage Gen2](#)
- **Performance tuning guidelines on Data Lake Storage Gen2**. For general performance concepts, see [Data Lake Storage Gen2 Performance Tuning Guidance](#)

## Parameters

When running Spark jobs, here are the most important settings that can be tuned to increase performance on Data Lake Storage Gen2:

- **Num-executors** - The number of concurrent tasks that can be executed.
- **Executor-memory** - The amount of memory allocated to each executor.
- **Executor-cores** - The number of cores allocated to each executor.

**Num-executors** Num-executors will set the maximum number of tasks that can run in parallel. The actual number of tasks that can run in parallel is bounded by the memory and CPU resources available in your cluster.

**Executor-memory** This is the amount of memory that is being allocated to each executor. The memory needed for each executor is dependent on the job. For complex operations, the memory needs to be higher. For simple operations like read and write, memory requirements will be lower. The amount of memory for each executor can be viewed in Ambari. In Ambari, navigate to Spark and view the Configs tab.

**Executor-cores** This sets the number of cores used per executor, which determines the number of parallel threads that can be run per executor. For example, if executor-cores = 2, then each executor can run 2 parallel tasks in the executor. The executor-cores needed will be dependent on the job. I/O heavy jobs do not require a large amount of memory per task so each executor can handle more parallel tasks.

By default, two virtual YARN cores are defined for each physical core when running Spark on HDInsight. This number provides a good balance of concurrency and amount of context switching from multiple threads.

## Guidance

While running Spark analytic workloads to work with data in Data Lake Storage Gen2, we recommend that you use the most recent HDInsight version to get the best performance with Data Lake Storage Gen2. When your job is

more I/O intensive, then certain parameters can be configured to improve performance. Data Lake Storage Gen2 is a highly scalable storage platform that can handle high throughput. If the job mainly consists of read or writes, then increasing concurrency for I/O to and from Data Lake Storage Gen2 could increase performance.

There are a few general ways to increase concurrency for I/O intensive jobs.

**Step 1: Determine how many apps are running on your cluster** – You should know how many apps are running on the cluster including the current one. The default values for each Spark setting assumes that there are 4 apps running concurrently. Therefore, you will only have 25% of the cluster available for each app. To get better performance, you can override the defaults by changing the number of executors.

**Step 2: Set executor-memory** – The first thing to set is the executor-memory. The memory will be dependent on the job that you are going to run. You can increase concurrency by allocating less memory per executor. If you see out of memory exceptions when you run your job, then you should increase the value for this parameter. One alternative is to get more memory by using a cluster that has higher amounts of memory or increasing the size of your cluster. More memory will enable more executors to be used, which means more concurrency.

**Step 3: Set executor-cores** – For I/O intensive workloads that do not have complex operations, it's good to start with a high number of executor-cores to increase the number of parallel tasks per executor. Setting executor-cores to 4 is a good start.

```
executor-cores = 4
```

Increasing the number of executor-cores will give you more parallelism so you can experiment with different executor-cores. For jobs that have more complex operations, you should reduce the number of cores per executor. If executor-cores is set higher than 4, then garbage collection may become inefficient and degrade performance.

**Step 4: Determine amount of YARN memory in cluster** – This information is available in Ambari. Navigate to YARN and view the Configs tab. The YARN memory is displayed in this window.

Note while you are in the window, you can also see the default YARN container size. The YARN container size is the same as memory per executor parameter.

```
Total YARN memory = nodes * YARN memory per node
```

## Step 5: Calculate num-executors

**Calculate memory constraint** - The num-executors parameter is constrained either by memory or by CPU. The memory constraint is determined by the amount of available YARN memory for your application. You should take total YARN memory and divide that by executor-memory. The constraint needs to be de-scaled for the number of apps so we divide by the number of apps.

```
Memory constraint = (total YARN memory / executor memory) / # of apps
```

**Calculate CPU constraint** - The CPU constraint is calculated as the total virtual cores divided by the number of cores per executor. There are 2 virtual cores for each physical core. Similar to the memory constraint, we have to divide by the number of apps.

```
virtual cores = (nodes in cluster * # of physical cores in node * 2)
CPU constraint = (total virtual cores / # of cores per executor) / # of apps
```

**Set num-executors** – The num-executors parameter is determined by taking the minimum of the memory constraint and the CPU constraint.

```
num-executors = Min (total virtual Cores / # of cores per executor, available YARN memory / executor-memory)
```

Setting a higher number of num-executors does not necessarily increase performance. You should consider that adding more executors will add extra overhead for each additional executor, which can potentially degrade performance. Num-executors is bounded by the cluster resources.

## Example calculation

Let's say you currently have a cluster composed of 8 D4v2 nodes that is running 2 apps including the one you are going to run.

**Step 1: Determine how many apps are running on your cluster** – you know that you have 2 apps on your cluster, including the one you are going to run.

**Step 2: Set executor-memory** – for this example, we determine that 6GB of executor-memory will be sufficient for I/O intensive job.

```
executor-memory = 6GB
```

**Step 3: Set executor-cores** – Since this is an I/O intensive job, we can set the number of cores for each executor to 4. Setting cores per executor to larger than 4 may cause garbage collection problems.

```
executor-cores = 4
```

**Step 4: Determine amount of YARN memory in cluster** – We navigate to Ambari to find out that each D4v2 has 25GB of YARN memory. Since there are 8 nodes, the available YARN memory is multiplied by 8.

```
Total YARN memory = nodes * YARN memory* per node
Total YARN memory = 8 nodes * 25GB = 200GB
```

**Step 5: Calculate num-executors** – The num-executors parameter is determined by taking the minimum of the memory constraint and the CPU constraint divided by the # of apps running on Spark.

**Calculate memory constraint** – The memory constraint is calculated as the total YARN memory divided by the memory per executor.

```
Memory constraint = (total YARN memory / executor memory) / # of apps
Memory constraint = (200GB / 6GB) / 2
Memory constraint = 16 (rounded)
```

**Calculate CPU constraint** - The CPU constraint is calculated as the total yarn cores divided by the number of cores per executor.

```
YARN cores = nodes in cluster * # of cores per node * 2
YARN cores = 8 nodes * 8 cores per D14 * 2 = 128
CPU constraint = (total YARN cores / # of cores per executor) / # of apps
CPU constraint = (128 / 4) / 2
CPU constraint = 16
```

## Set num-executors

```
num-executors = Min (memory constraint, CPU constraint)
num-executors = Min (16, 16)
num-executors = 16
```

# Tune performance: Hive, HDInsight & Azure Data Lake Storage Gen2

11/21/2019 • 3 minutes to read • [Edit Online](#)

The default settings have been set to provide good performance across many different use cases. For I/O intensive queries, Hive can be tuned to get better performance with Azure Data Lake Storage Gen2.

## Prerequisites

- An Azure subscription. See [Get Azure free trial](#).
- A **Data Lake Storage Gen2 account**. For instructions on how to create one, see [Quickstart: Create an Azure Data Lake Storage Gen2 storage account](#)
- Azure **HDInsight cluster** with access to a Data Lake Storage Gen2 account. See [Use Azure Data Lake Storage Gen2 with Azure HDInsight clusters](#)
- Running **Hive on HDInsight**. To learn about running Hive jobs on HDInsight, see [Use Hive on HDInsight](#)
- **Performance tuning guidelines on Data Lake Storage Gen2**. For general performance concepts, see [Data Lake Storage Gen2 Performance Tuning Guidance](#)

## Parameters

Here are the most important settings to tune for improved Data Lake Storage Gen2 performance:

- **hive.tez.container.size** – the amount of memory used by each tasks
- **tez.grouping.min-size** – minimum size of each mapper
- **tez.grouping.max-size** – maximum size of each mapper
- **hive.exec.reducer.bytes.per.reducer** – size of each reducer

**hive.tez.container.size** - The container size determines how much memory is available for each task. This is the main input for controlling the concurrency in Hive.

**tez.grouping.min-size** – This parameter allows you to set the minimum size of each mapper. If the number of mappers that Tez chooses is smaller than the value of this parameter, then Tez will use the value set here.

**tez.grouping.max-size** – The parameter allows you to set the maximum size of each mapper. If the number of mappers that Tez chooses is larger than the value of this parameter, then Tez will use the value set here.

**hive.exec.reducer.bytes.per.reducer** – This parameter sets the size of each reducer. By default, each reducer is 256MB.

## Guidance

Set **hive.exec.reducer.bytes.per.reducer** – The default value works well when the data is uncompressed. For data that is compressed, you should reduce the size of the reducer.

Set **hive.tez.container.size** – In each node, memory is specified by `yarn.nodemanager.resource.memory-mb` and should be correctly set on HDI cluster by default. For additional information on setting the appropriate memory in YARN, see this [post](#).

I/O intensive workloads can benefit from more parallelism by decreasing the Tez container size. This gives the user

more containers which increases concurrency. However, some Hive queries require a significant amount of memory (e.g. MapJoin). If the task does not have enough memory, you will get an out of memory exception during runtime. If you receive out of memory exceptions, then you should increase the memory.

The concurrent number of tasks running or parallelism will be bounded by the total YARN memory. The number of YARN containers will dictate how many concurrent tasks can run. To find the YARN memory per node, you can go to Ambari. Navigate to YARN and view the Configs tab. The YARN memory is displayed in this window.

```
Total YARN memory = nodes * YARN memory per node
of YARN containers = Total YARN memory / Tez container size
```

The key to improving performance using Data Lake Storage Gen2 is to increase the concurrency as much as possible. Tez automatically calculates the number of tasks that should be created so you do not need to set it.

## Example calculation

Let's say you have an 8 node D14 cluster.

```
Total YARN memory = nodes * YARN memory per node
Total YARN memory = 8 nodes * 96GB = 768GB
of YARN containers = 768GB / 3072MB = 256
```

## Further information on Hive tuning

Here are a few blogs that will help tune your Hive queries:

- [Optimize Hive queries for Hadoop in HDInsight](#)
- [Troubleshooting Hive query performance](#)
- [Ignite talk on optimize Hive on HDInsight](#)

# Tune performance: MapReduce, HDInsight & Azure Data Lake Storage Gen2

11/21/2019 • 4 minutes to read • [Edit Online](#)

Understand the factors that you should consider when you tune the performance of Map Reduce jobs. This article covers a range of performance tuning guidelines.

## Prerequisites

- An Azure subscription. See [Get Azure free trial](#).
- An **Azure Data Lake Storage Gen2 account**. For instructions on how to create one, see [Quickstart: Create an Azure Data Lake Storage Gen2 storage account](#).
- **Azure HDInsight cluster** with access to a Data Lake Storage Gen2 account. See [Use Azure Data Lake Storage Gen2 with Azure HDInsight clusters](#)
- **Using MapReduce on HDInsight**. For more information, see [Use MapReduce in Hadoop on HDInsight](#)
- **Performance tuning guidelines on Data Lake Storage Gen2**. For general performance concepts, see [Data Lake Storage Gen2 Performance Tuning Guidance](#)

## Parameters

When running MapReduce jobs, here are the parameters that you can configure to increase performance on Data Lake Storage Gen2:

- **Mapreduce.map.memory.mb** – The amount of memory to allocate to each mapper
- **Mapreduce.job.maps** – The number of map tasks per job
- **Mapreduce.reduce.memory.mb** – The amount of memory to allocate to each reducer
- **Mapreduce.job.reduces** – The number of reduce tasks per job

**Mapreduce.map.memory / Mapreduce.reduce.memory** This number should be adjusted based on how much memory is needed for the map and/or reduce task. The default values of mapreduce.map.memory and mapreduce.reduce.memory can be viewed in Ambari via the Yarn configuration. In Ambari, navigate to YARN and view the Configs tab. The YARN memory will be displayed.

**Mapreduce.job.maps / Mapreduce.job.reduces** This will determine the maximum number of mappers or reducers to be created. The number of splits will determine how many mappers will be created for the MapReduce job. Therefore, you may get less mappers than you requested if there are less splits than the number of mappers requested.

## Guidance

### NOTE

The guidance in this document assumes that your application is the only application running on your cluster.

### Step 1: Determine number of jobs running

By default, MapReduce will use the entire cluster for your job. You can use less of the cluster by using less mappers than there are available containers.

## **Step 2: Set mapreduce.map.memory/mapreduce.reduce.memory**

The size of the memory for map and reduce tasks will be dependent on your specific job. You can reduce the memory size if you want to increase concurrency. The number of concurrently running tasks depends on the number of containers. By decreasing the amount of memory per mapper or reducer, more containers can be created, which enable more mappers or reducers to run concurrently. Decreasing the amount of memory too much may cause some processes to run out of memory. If you get a heap error when running your job, you should increase the memory per mapper or reducer. You should consider that adding more containers will add extra overhead for each additional container, which can potentially degrade performance. Another alternative is to get more memory by using a cluster that has higher amounts of memory or increasing the number of nodes in your cluster. More memory will enable more containers to be used, which means more concurrency.

## **Step 3: Determine Total YARN memory**

To tune mapreduce.job.maps/mapreduce.job.reduces, you should consider the amount of total YARN memory available for use. This information is available in Ambari. Navigate to YARN and view the Configs tab. The YARN memory is displayed in this window. You should multiply the YARN memory with the number of nodes in your cluster to get the total YARN memory.

```
Total YARN memory = nodes * YARN memory per node
```

If you are using an empty cluster, then memory can be the total YARN memory for your cluster. If other applications are using memory, then you can choose to only use a portion of your cluster's memory by reducing the number of mappers or reducers to the number of containers you want to use.

## **Step 4: Calculate number of YARN containers**

YARN containers dictate the amount of concurrency available for the job. Take total YARN memory and divide that by mapreduce.map.memory.

```
of YARN containers = total YARN memory / mapreduce.map.memory
```

## **Step 5: Set mapreduce.job.maps/mapreduce.job.reduces**

Set mapreduce.job.maps/mapreduce.job.reduces to at least the number of available containers. You can experiment further by increasing the number of mappers and reducers to see if you get better performance. Keep in mind that more mappers will have additional overhead so having too many mappers may degrade performance.

CPU scheduling and CPU isolation are turned off by default so the number of YARN containers is constrained by memory.

# **Example calculation**

Let's assume that we have a cluster composed of 8 D14 nodes, and we want to run an I/O intensive job. Here are the calculations you should do:

## **Step 1: Determine number of jobs running**

In this example, let's assume that our job is the only job that is running.

## **Step 2: Set mapreduce.map.memory/mapreduce.reduce.memory**

In this example, we are running an I/O intensive job and decide that 3GB of memory for map tasks will be sufficient.

```
mapreduce.map.memory = 3GB
```

### Step 3: Determine Total YARN memory

```
Total memory from the cluster is 8 nodes * 96GB of YARN memory for a D14 = 768GB
```

### Step 4: Calculate # of YARN containers

```
of YARN containers = 768GB of available memory / 3 GB of memory = 256
```

### Step 5: Set mapreduce.job.maps/mapreduce.job.reduces

```
mapreduce.map.jobs = 256
```

## Examples to run

To demonstrate how MapReduce runs on Data Lake Storage Gen2, below is some sample code that was run on a cluster with the following settings:

- 16 node D14v2
- Hadoop cluster running HDI 3.6

For a starting point, here are some example commands to run MapReduce Teragen, Terasort, and Teravalidate. You can adjust these commands based on your resources.

### Teragen

```
yarn jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-mapreduce-examples.jar teragen -Dmapreduce.job.maps=2048 -Dmapreduce.map.memory.mb=3072 10000000000 abfs://example/data/1TB-sort-input
```

### Terasort

```
yarn jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-mapreduce-examples.jar terasort -Dmapreduce.job.maps=2048 -Dmapreduce.map.memory.mb=3072 -Dmapreduce.job.reduces=512 -Dmapreduce.reduce.memory.mb=3072 abfs://example/data/1TB-sort-input abfs://example/data/1TB-sort-output
```

### Teravalidate

```
yarn jar /usr/hdp/current/hadoop-mapreduce-client/hadoop-mapreduce-examples.jar teravalidate -Dmapreduce.job.maps=512 -Dmapreduce.map.memory.mb=3072 abfs://example/data/1TB-sort-output abfs://example/data/1TB-sort-validate
```

# Tune performance: Storm, HDInsight & Azure Data Lake Storage Gen2

11/21/2019 • 8 minutes to read • [Edit Online](#)

Understand the factors that should be considered when you tune the performance of an Azure Storm topology. For example, it's important to understand the characteristics of the work done by the spouts and the bolts (whether the work is I/O or memory intensive). This article covers a range of performance tuning guidelines, including troubleshooting common issues.

## Prerequisites

- An Azure subscription. See [Get Azure free trial](#).
- An Azure Data Lake Storage Gen2 account. For instructions on how to create one, see [Quickstart: Create a storage account for analytic](#).
- Azure HDInsight cluster with access to a Data Lake Storage Gen2 account. See [Use Azure Data Lake Storage Gen2 with Azure HDInsight clusters](#). Make sure you enable Remote Desktop for the cluster.
- Running a Storm cluster on Data Lake Storage Gen2. For more information, see [Storm on HDInsight](#).
- Performance tuning guidelines on Data Lake Storage Gen2. For general performance concepts, see [Data Lake Storage Gen2 Performance Tuning Guidance](#).

## Tune the parallelism of the topology

You might be able to improve performance by increasing the concurrency of the I/O to and from Data Lake Storage Gen2. A Storm topology has a set of configurations that determine the parallelism:

- Number of worker processes (the workers are evenly distributed across the VMs).
- Number of spout executor instances.
- Number of bolt executor instances.
- Number of spout tasks.
- Number of bolt tasks.

For example, on a cluster with 4 VMs and 4 worker processes, 32 spout executors and 32 spout tasks, and 256 bolt executors and 512 bolt tasks, consider the following:

Each supervisor, which is a worker node, has a single worker Java virtual machine (JVM) process. This JVM process manages 4 spout threads and 64 bolt threads. Within each thread, tasks are run sequentially. With the preceding configuration, each spout thread has 1 task, and each bolt thread has 2 tasks.

In Storm, here are the various components involved, and how they affect the level of parallelism you have:

- The head node (called Nimbus in Storm) is used to submit and manage jobs. These nodes have no impact on the degree of parallelism.
- The supervisor nodes. In HDInsight, this corresponds to a worker node Azure VM.
- The worker tasks are Storm processes running in the VMs. Each worker task corresponds to a JVM instance. Storm distributes the number of worker processes you specify to the worker nodes as evenly as possible.
- Spout and bolt executor instances. Each executor instance corresponds to a thread running within the workers (JVMs).
- Storm tasks. These are logical tasks that each of these threads run. This does not change the level of parallelism, so you should evaluate if you need multiple tasks per executor or not.

## Get the best performance from Data Lake Storage Gen2

When working with Data Lake Storage Gen2, you get the best performance if you do the following:

- Coalesce your small appends into larger sizes.
- Do as many concurrent requests as you can. Because each bolt thread is doing blocking reads, you want to have somewhere in the range of 8-12 threads per core. This keeps the NIC and the CPU well utilized. A larger VM enables more concurrent requests.

### Example topology

Let's assume you have an 8 worker node cluster with a D13v2 Azure VM. This VM has 8 cores, so among the 8 worker nodes, you have 64 total cores.

Let's say we do 8 bolt threads per core. Given 64 cores, that means we want 512 total bolt executor instances (that is, threads). In this case, let's say we start with one JVM per VM, and mainly use the thread concurrency within the JVM to achieve concurrency. That means we need 8 worker tasks (one per Azure VM), and 512 bolt executors.

Given this configuration, Storm tries to distribute the workers evenly across worker nodes (also known as supervisor nodes), giving each worker node 1 JVM. Now within the supervisors, Storm tries to distribute the executors evenly between supervisors, giving each supervisor (that is, JVM) 8 threads each.

## Tune additional parameters

After you have the basic topology, you can consider whether you want to tweak any of the parameters:

- **Number of JVMs per worker node.** If you have a large data structure (for example, a lookup table) that you host in memory, each JVM requires a separate copy. Alternatively, you can use the data structure across many threads if you have fewer JVMs. For the bolt's I/O, the number of JVMs does not make as much of a difference as the number of threads added across those JVMs. For simplicity, it's a good idea to have one JVM per worker. Depending on what your bolt is doing or what application processing you require, though, you may need to change this number.
- **Number of spout executors.** Because the preceding example uses bolts for writing to Data Lake Storage Gen2, the number of spouts is not directly relevant to the bolt performance. However, depending on the amount of processing or I/O happening in the spout, it's a good idea to tune the spouts for best performance. Ensure that you have enough spouts to be able to keep the bolts busy. The output rates of the spouts should match the throughput of the bolts. The actual configuration depends on the spout.
- **Number of tasks.** Each bolt runs as a single thread. Additional tasks per bolt don't provide any additional concurrency. The only time they are of benefit is if your process of acknowledging the tuple takes a large proportion of your bolt execution time. It's a good idea to group many tuples into a larger append before you send an acknowledgement from the bolt. So, in most cases, multiple tasks provide no additional benefit.
- **Local or shuffle grouping.** When this setting is enabled, tuples are sent to bolts within the same worker process. This reduces inter-process communication and network calls. This is recommended for most topologies.

This basic scenario is a good starting point. Test with your own data to tweak the preceding parameters to achieve optimal performance.

## Tune the spout

You can modify the following settings to tune the spout.

- **Tuple timeout: topology.message.timeout.secs.** This setting determines the amount of time a message takes to complete, and receive acknowledgement, before it is considered failed.
- **Max memory per worker process: worker.childopts.** This setting lets you specify additional command-line parameters to the Java workers. The most commonly used setting here is XmX, which determines the maximum memory allocated to a JVM's heap.

- **Max spout pending:** `topology.max.spout.pending`. This setting determines the number of tuples that can be in flight (not yet acknowledged at all nodes in the topology) per spout thread at any time.

A good calculation to do is to estimate the size of each of your tuples. Then figure out how much memory one spout thread has. The total memory allocated to a thread, divided by this value, should give you the upper bound for the max spout pending parameter.

The default Data Lake Storage Gen2 Storm bolt has a size sync policy parameter (`fileBufferSize`) that can be used to tune this parameter.

In I/O-intensive topologies, it's a good idea to have each bolt thread write to its own file, and to set a file rotation policy (`fileRotationSize`). When the file reaches a certain size, the stream is automatically flushed and a new file is written to. The recommended file size for rotation is 1 GB.

## Monitor your topology in Storm

While your topology is running, you can monitor it in the Storm user interface. Here are the main parameters to look at:

- **Total process execution latency.** This is the average time one tuple takes to be emitted by the spout, processed by the bolt, and acknowledged.
- **Total bolt process latency.** This is the average time spent by the tuple at the bolt until it receives an acknowledgement.
- **Total bolt execute latency.** This is the average time spent by the bolt in the `execute` method.
- **Number of failures.** This refers to the number of tuples that failed to be fully processed before they timed out.
- **Capacity.** This is a measure of how busy your system is. If this number is 1, your bolts are working as fast as they can. If it is less than 1, increase the parallelism. If it is greater than 1, reduce the parallelism.

## Troubleshoot common problems

Here are a few common troubleshooting scenarios.

- **Many tuples are timing out.** Look at each node in the topology to determine where the bottleneck is. The most common reason for this is that the bolts are not able to keep up with the spouts. This leads to tuples clogging the internal buffers while waiting to be processed. Consider increasing the timeout value or decreasing the max spout pending.
- **There is a high total process execution latency, but a low bolt process latency.** In this case, it is possible that the tuples are not being acknowledged fast enough. Check that there are a sufficient number of acknowledgers. Another possibility is that they are waiting in the queue for too long before the bolts start processing them. Decrease the max spout pending.
- **There is a high bolt execute latency.** This means that the `execute()` method of your bolt is taking too long. Optimize the code, or look at write sizes and flush behavior.

### Data Lake Storage Gen2 throttling

If you hit the limits of bandwidth provided by Data Lake Storage Gen2, you might see task failures. Check task logs for throttling errors. You can decrease the parallelism by increasing container size.

To check if you are getting throttled, enable the debug logging on the client side:

1. In Ambari > Storm > Config > Advanced `storm-worker-log4j`, change `<root level="info">` to `<root level="debug">`. Restart all the nodes/service for the configuration to take effect.

2. Monitor the Storm topology logs on worker nodes (under /var/log/storm/worker-artifacts/<TopologyName>/<port>/worker.log) for Data Lake Storage Gen2 throttling exceptions.

## Next steps

Additional performance tuning for Storm can be referenced in [this blog](#).

For an additional example to run, see [this one on GitHub](#).

# Check the Last Sync Time property for a storage account

2/12/2020 • 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) (preview) 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 an Azure Storage preview module that supports getting geo-replication stats. For example:

```
Install-Module Az.Storage -Repository PSGallery -RequiredVersion 1.1.1-preview -AllowPrerelease -AllowClobber -Force
```

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](#)
- [Designing highly available applications using read-access geo-redundant storage](#)

# Use the Azurite emulator for local Azure Storage development and testing (preview)

1/15/2020 • 9 minutes to read • [Edit Online](#)

The Azurite version 3.2 open-source emulator (preview) provides a free local environment for testing your Azure blob and queue 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 v3 supports APIs implemented by the Azure Blob service.

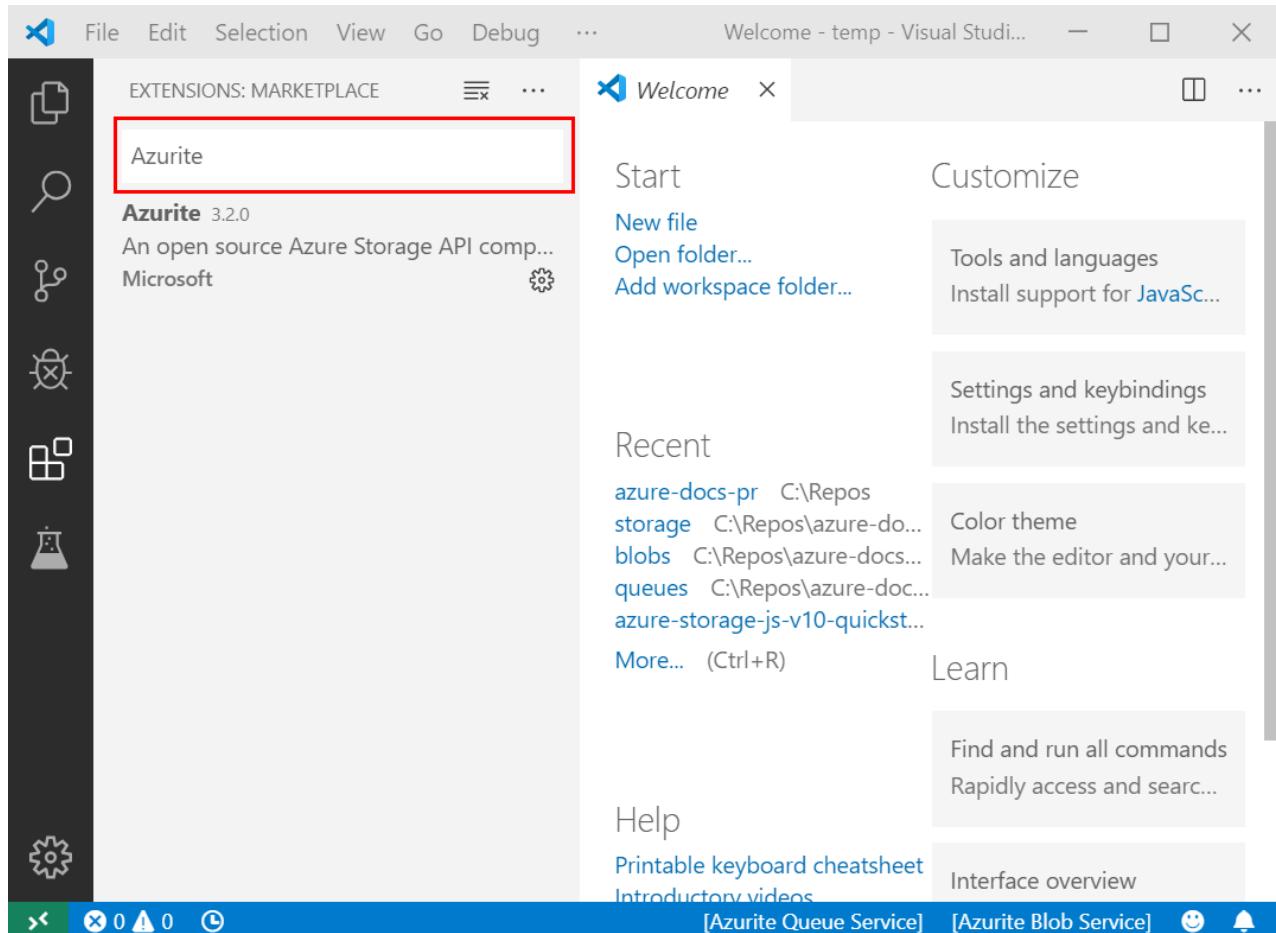
Azurite is the future storage emulator platform. Azurite supersedes the [Azure Storage Emulator](#). Azurite will continue to be updated to support the latest versions of Azure Storage APIs.

There are several different ways to install and run Azurite on your local system:

1. [Install and run the Azurite Visual Studio Code extension](#)
2. [Install and run Azurite by using NPM](#)
3. [Install and run the Azurite Docker image](#)
4. [Clone, build, and run Azurite from the GitHub repository](#)

## Install and run the Azurite Visual Studio Code extension

Within Visual Studio Code, select the EXTENSIONS pane and search for *Azurite* in the EXTENSIONS:MARKETPLACE.



Alternatively, navigate to [VS Code extension market](#) in your browser. Select the **Install** button to open Visual Studio

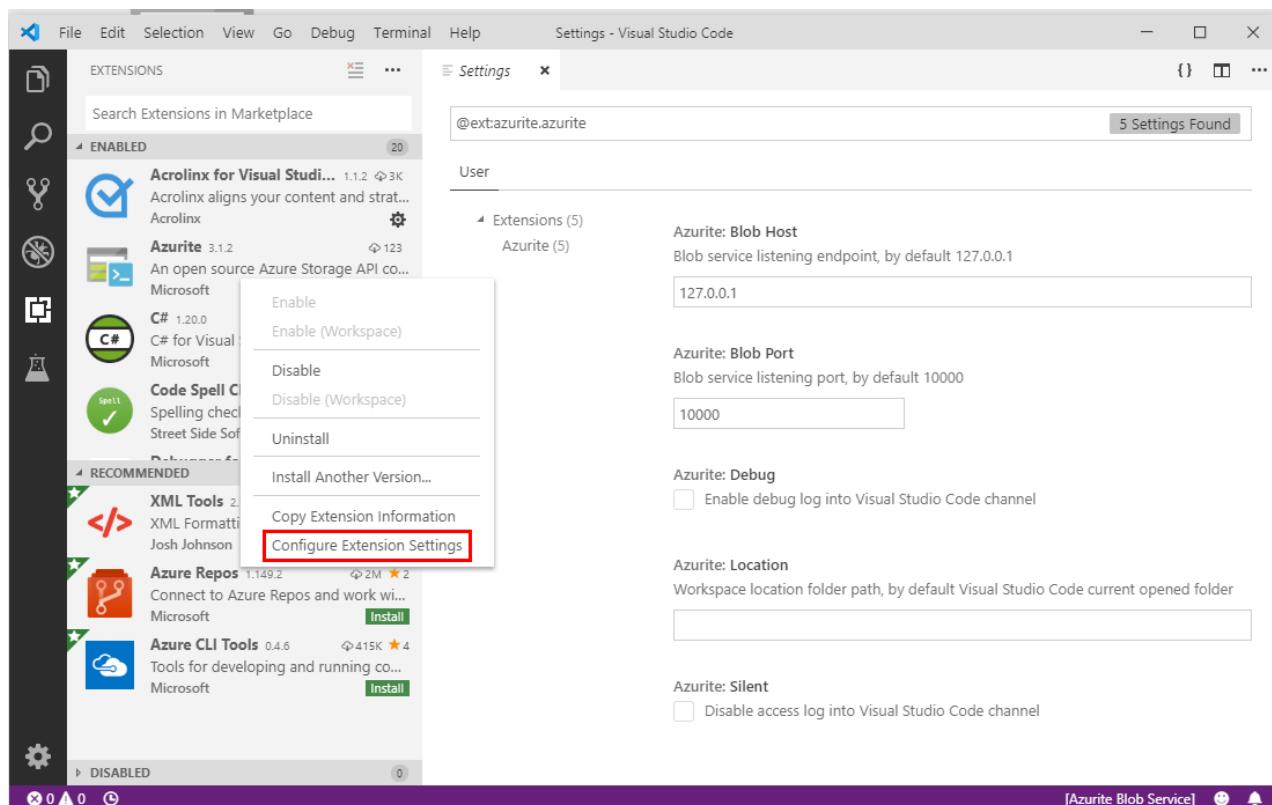
Code and go directly to the Azurite extension page.

You can quickly start or close Azurite by clicking on [Azurite Blob Service] or [Azurite Queue Service] in the VS Code status bar or issuing the following commands in the VS Code command palette. To open the command palette, press F1 in VS Code.

The extension supports the following Visual Studio Code commands:

- **Azurite: Start** - Start all Azurite services
- **Azurite: Close** - Close all Azurite services
- **Azurite: Clean** - Reset all Azurite services persistency data
- **Azurite: Start Blob Service** - Start blob service
- **Azurite: Close Blob Service** - Close blob service
- **Azurite: Clean Blob Service** - Clean blob service
- **Azurite: Start Queue Service** - Start queue service
- **Azurite: Close Queue Service** - Close queue service
- **Azurite: Clean Queue Service** - Clean queue service

To configure Azurite within Visual Studio Code, select the extensions pane. Select the **Manage** (gear) icon for **Azurite**. Select **Configure Extension Settings**.



The following settings are supported:

- **Azurite: Blob Host** - The Blob service listening endpoint. The default setting is 127.0.0.1.
- **Azurite: Blob Port** - The Blob service listening port. The default port is 10000.
- **Azurite: Debug** - Output the debug log to the Azurite channel. The default value is **false**.
- **Azurite: Location** - The workspace location path. The default is the Visual Studio Code working folder.
- **Azurite: Queue Host** - The Queue service listening endpoint. The default setting is 127.0.0.1.
- **Azurite: Queue Port** - The Queue service listening port. The default port is 10001.
- **Azurite: Silent** - Silent mode disables the access log. The default value is **false**.

## Install and run Azurite by using NPM

This installation method requires that you have [Node.js version 8.0 or later](#) installed. **npm** is the package management tool included with every Node.js installation. After installing Node.js, execute the following **npm** command to install Azurite.

```
npm install -g azurite
```

After installing Azurite, see [Run Azurite from a command-line](#).

## Install and run the Azurite Docker image

Use [DockerHub](#) to pull the [latest Azurite image](#) by using the following command:

```
docker pull mcr.microsoft.com/azure-storage/azurite
```

### Run the Azurite Docker image:

The following command runs the Azurite Docker image. The `-p 10000:10000` parameter redirects requests from host machine's port 10000 to the Docker instance.

```
docker run -p 10000:10000 -p 10001:10001 mcr.microsoft.com/azure-storage/azurite
```

### Specify the workspace location:

In the following example, the `-v c:/azurite:/data` parameter specifies *c:/azurite* as the Azurite persisted data location. The directory, *c:/azurite*, must be created before running the Docker command.

```
docker run -p 10000:10000 -p 10001:10001 -v c:/azurite:/data mcr.microsoft.com/azure-storage/azurite
```

### Run just the blob service

```
docker run -p 10000:10000 mcr.microsoft.com/azure-storage/azurite
azurite-blob --blobHost 0.0.0.0 --blobPort 10000
```

### Set all Azurite parameters:

This example shows how to set all of the command-line parameters. All of the parameters below should be placed on a single command-line.

```
docker run -p 8888:8888
-p 9999:9999
-v c:/azurite:/workspace mcr.microsoft.com/azure-storage/azurite azurite
-l /workspace
-d /workspace/debug.log
--blobPort 8888
--blobHost 0.0.0.0
--queuePort 9999
--queueHost 0.0.0.0
```

See [Command-line options](#) for more information about configuring Azurite at start-up.

## Clone, build, and run Azurite from the GitHub repository

This installation method requires that you have [Git](#) installed. Clone the [GitHub repository](#) for the Azurite project by

using the following console command.

```
git clone https://github.com/Azure/Azurite.git
```

After cloning the source code, execute following commands from the root of the cloned repo to build and install Azurite.

```
npm install
npm run build
npm install -g
```

After installing and building Azurite, see [Run Azurite from a command-line](#).

## Run Azurite from a command-line

### NOTE

Azurite cannot be run from the command line if you only installed the Visual Studio Code extension. Instead, use the VS Code command palette. For more information, see [Install and run the Azurite Visual Studio Code extension](#).

To get started immediately with the command-line, create a directory called `c:\azurite`, then launch Azurite by issuing the following command:

```
azurite --silent --location c:\azurite --debug c:\azurite\debug.log
```

This command tells Azurite to store all data in a particular directory, `c:\azurite`. If the `--location` option is omitted, it will use the current working directory.

## Command-line options

This section details the command-line switches available when launching Azurite. All command-line switches are optional.

```
C:\Azurite> azurite [--blobHost <IP address>] [--blobPort <port address>]
[-d | --debug <log file path>] [-l | --location <workspace path>]
[--queueHost <IP address>] [--queuePort <port address>]
[-s | --silent] [-h | --help]
```

The `-d` is a shortcut for `--debug`, `-l` switch is a shortcut for `--location`, `-s` is a shortcut for `--silent`, and `-h` is a shortcut for `--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.

## Workspace path

**Optional** Azurite stores data to the local disk during execution. Use the **--location** switch to specify a path as the workspace location. By default, the current process working directory will be used.

```
azurite --location c:\azurite
```

```
azurite -l 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 **--silent** switch.

```
azurite --silent
```

```
azurite -s
```

## 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 **--debug** switch.

```
azurite --debug path/debug.log
```

```
azurite -d 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 **--loose** switch.

```
azurite --loose
```

Note the capital 'L' shortcut switch:

```
azurite -L
```

## 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 Shared Key and shared access signatures (SAS). Azurite also supports anonymous access to public containers.

## Well-known storage account and key

You can use the following account name and key with Azurite. This is the same well-known account and key used by the legacy Azure storage emulator.

- Account name: `devstoreaccount1`
- Account key: `Eby8vdM02xNOcqFlqUwJPLlmEt1CDXJ10UzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==`

### NOTE

In addition to SharedKey authentication, Azurite supports account and service SAS authentication. Anonymous access is also available when a container is set to allow public access.

## Connection string

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>
```

For more information, see [Configure Azure Storage connection strings](#).

## 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"
```

Or use multiple storage accounts with 2 keys each:

```
set 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.

### NOTE

Update the connection string accordingly when using custom account names and keys.

## NOTE

Use the `export` keyword to set environment variables in a Linux environment, use `set` in Windows.

## Storage Explorer

In Azure Storage Explorer, connect to Azurite by clicking the **Add Account** icon, then select **Attach to a local emulator** and click **Connect**.

## 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
```

Since 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 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
```

### Scaling and performance

Azurite is not a scalable storage service and does not support a large number of concurrent 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
```

## 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.

# Use the Azure storage emulator for development and testing

3/30/2020 • 17 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.

## 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. If you need a storage emulator for Linux, one option is the community maintained, open-source storage emulator [Azurite](#).

### 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 reinitializes 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 storage 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 storage emulator. They are:

```
Account name: devstoreaccount1
Account key: Eby8vdM02xN0cqFlqUwJPLlmEt1CDXJ1OUzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==
```

### NOTE

The authentication key supported by the storage 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 storage emulator. You should not use the development account with production data.

The storage 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 a shortcut

The easiest way to connect to the storage emulator 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 to the storage emulator in an *app.config* file:

```
<appSettings>
 <add key="StorageConnectionString" value="UseDevelopmentStorage=true" />
</appSettings>
```

## Connect to the emulator account using the well-known account name and key

To create a connection string that references the emulator account name and key, you must specify the endpoints for each of the services you wish to use from the emulator in the connection string. This is necessary so that the connection string will reference the emulator endpoints, which are different than those for a production storage account. For example, the value of your connection string will look like this:

```
DefaultEndpointsProtocol=http;AccountName=devstoreaccount1;
AccountKey=Eby8vdM02xN0cqFlqUwJPLlmEt1CDXJ1OUzFT50uSRZ6IFsuFq2UVErCz4I6tq/K1SZFPT0tr/KBHBeksoGMGw==;
BlobEndpoint=http://127.0.0.1:10000/devstoreaccount1;
TableEndpoint=http://127.0.0.1:10002/devstoreaccount1;
QueueEndpoint=http://127.0.0.1:10001/devstoreaccount1;
```

This value is identical to the shortcut shown above, `UseDevelopmentStorage=true`.

## Specify an HTTP proxy

You can also specify an HTTP proxy to use when you're testing your service against the storage emulator. This can be useful for observing HTTP requests and responses while you're debugging operations against the storage services. To specify a proxy, add the `DevelopmentStorageProxyUri` option to the connection string, and set its value to the proxy URI. For example, here is a connection string that points to the storage emulator and configures an HTTP proxy:

```
UseDevelopmentStorage=true;DevelopmentStorageProxyUri=http://myProxyUri
```

For more information on connection strings, see [Configure Azure Storage connection strings](#).

## Authorize with a shared access signature

### NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

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](http://<local-machine-address>:<port>/<account-name>/<resource-path>) at the command prompt.

OPTION	DESCRIPTION	COMMAND	ARGUMENTS
Start	Starts up the storage emulator.	<a href="http://&lt;local-machine-address&gt;:&lt;port&gt;/&lt;account-name&gt;/&lt;resource-path&gt;">AzureStorageEmulator.exe start [-inprocess]</a>	-Reprocess: Start the emulator in the current process instead of creating a new process.

OPTION	DESCRIPTION	COMMAND	ARGUMENTS
Stop	Stops the storage emulator.	AzureStorageEmulator.exe stop	
Status	Prints the status of the storage emulator.	AzureStorageEmulator.exe status	
Clear	Clears the data in all services specified on the command line.	AzureStorageEmulator.exe clear [blob] [table] [queue] [all]	<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 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 VersionNotSupportedException 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.

# Query acceleration SQL language reference (preview)

4/21/2020 • 8 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.

## NOTE

The query acceleration feature is in public preview, and is available in the Canada Central and France Central regions. To review limitations, see the [Known issues](#) article. To enroll in the preview, see [this form](#).

## 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]
```

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.

DATA TYPE	DESCRIPTION
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:

=, !=, <>, <, <=, >, >=, +, -, /, \*, %, AND, OR, NOT, CAST, BETWEEN, IN, NULLIF, COALESCE

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:

LIKE, CHAR\_LENGTH, CHARACTER\_LENGTH, LOWER, UPPER, SUBSTRING, TRIM, LEADING, TRAILING.

Here's a few examples:

FUNCTION	EXAMPLE	RESULT
CHARACTER_LENGTH	SELECT CHARACTER_LENGTH('abcdefg') from BlobStorage	7

FUNCTION	EXAMPLE	RESULT
CHAR_LENGTH	SELECT CHAR_LENGTH(_1) from BlobStorage	1
LOWER	SELECT LOWER('AbCdEfG') from BlobStorage	abcdefg
UPPER	SELECT UPPER('AbCdEfG') from BlobStorage	ABCDEFG
SUBSTRING	SUBSTRING('123456789', 1, 5)	23456
TRIM	TRIM(BOTH '123' FROM '11122211Microsoft22211122')	Microsoft

The [LIKE](#) function helps you to search for a pattern. Here's a few examples that use the [LIKE](#) function to search the data string `abc,abd,cd\n test,test2,test3\na_bc,xc%d^e,gh[i]`.

QUERY	EXAMPLE
SELECT _1, _2, _3 from BlobStorage where _2 LIKE 'a%'	abc,abd,cd\n
SELECT * from BlobStorage where _1 LIKE 'a[bcd]c'	abc,abd,cd\n
SELECT _1 from BlobStorage where _2 LIKE '[^xyz]%'	abc\n test\n
SELECT * from BlobStorage where _1 LIKE 'a_	abc,abd,cd\n
SELECT _2,_3 from BlobStorage where _3 LIKE '[g-h]_! [[a-j]]' Escape '!''	xc%d^e,gh[i\n

## Date functions

The following standard SQL date functions are supported:

[DATE\\_ADD](#), [DATE\\_DIFF](#), [EXTRACT](#), [TO\\_STRING](#), [TO\\_TIMESTAMP](#).

Currently we convert all the [date formats of standard ISO8601](#).

### 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
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)

FORMAT STRING	OUTPUT
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>AVERAGE(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.

EXPRESSION	DESCRIPTION
SUM(expression)	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, you're 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 \(preview\)](#)
- [Filter data by using Azure Data Lake Storage query acceleration \(preview\)](#)

# azcopy

11/13/2019 • 2 minutes to read • [Edit Online](#)

AzCopy is a command-line tool that moves data into and out of Azure Storage.

## 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](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Options

**--cap-mbps uint32** 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** Format of the command's output. The choices include: text, json. The default value is 'text'. (default "text")

## 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 logout](#)
- [azcopy make](#)
- [azcopy remove](#)
- [azcopy sync](#)

# azcopy bench

10/16/2019 • 2 minutes to read • [Edit Online](#)

Runs a performance benchmark by uploading test data to a specified destination. The test data is automatically generated.

The benchmark command runs the same upload process as 'copy', except that:

- There's no source parameter. The command requires only a destination URL. In the current release, this destination URL must refer to a blob container.
- 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 is not used.
- Only a few of the optional parameters that are available to the copy command are supported.
- Additional diagnostics are measured and reported.
- By default, the transferred data is deleted at the end of the test run.

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 AZCOPY\_CONCURRENCY\_VALUE environment variable to a specific number of connections.

All the usual authentication types are supported. However, the most convenient approach for benchmarking is typically to create an empty container with a SAS token and use SAS authentication.

## Examples

```
azcopy bench [destination] [flags]
```

Run a benchmark test with default parameters (suitable for benchmarking networks up to 1 Gbps):'

- azcopy bench "https://[account].blob.core.windows.net/[container]?"

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]?" --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 "https://[account].blob.core.windows.net/[container]?" --file-count 50000 --size-per-file 8M --put-md5

## 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 - e.g. 0.25. Identical to the same-named parameter in the copy command.

**--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 - e.g. to use it for manual tests outside benchmark mode (default true).

**--file-count** uint The 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")

**--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** uint32 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").

## See also

- [azcopy](#)

# azcopy copy

4/10/2020 • 10 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 <-> ADLS Gen 2 (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)

Please refer to the examples for more information.

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## 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 is 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. Please 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 have not 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]"

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

Download a single file by using OAuth authentication. If you have not 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]" > "/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

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. You have to use a SAS token at the end of the source account URL, but the destination account doesn't need one if you log into AzCopy by using the azcopy login command.

- azcopy cp "https://[srcaccount].blob.core.windows.net/[container]/[path/to/blob]?[SAS]" "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

Please 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

## Options

**--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 (e.g. has administrator rights or is a member of the 'Backup Operators' group). All this flag does is activate privileges that the account already has.

**--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 blobs directly to the [access tier](#) of your choice. (default 'None'). Valid values include 'None', 'Hot', 'Cool', and 'Archive'. If 'None' or no tier is passed, the blob will inherit the tier of the storage account.

**--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 is 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")

**--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.

**--decompress** Automatically decompress files when downloading, if their content-encoding indicates that they are 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.

**--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 is not 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 does not support wildcard characters (\*). Checks relative path prefix(For example: myFolder;myFolder/subDirName/file.pdf). When used in combination with account traversal, paths do not include the container name.

**--exclude-pattern** string Exclude these files when copying. This option supports wildcard characters (\*)

**--follow-symlinks** Follow symbolic links when uploading from local file system.

**--from-to** string Optionally specifies the source destination combination. For Example: LocalBlob, BlobLocal, LocalBlobFS.

**-h, --help** help for copy

**--include-attributes** string (Windows only) Include files whose attributes match the attribute list. For example: A;S;R

**--include-path** string Include only these paths when copying. This option does not 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 ':'.

**--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")

**--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. Possible values include 'true', 'false', 'ifSourceNewer', and 'prompt'. (default "true")

**--page-blob-tier** string Upload page blob to Azure Storage using this blob tier. (default "None")

**--preserve-last-modified-time** Only available when destination is file system.

**--preserve-smb-permissions** string False by default. Preserves SMB ACLs between aware resources (Windows and Azure Files). For downloads, you will also need to use the **--backup** flag to restore permissions where the new Owner will not be the user that is running AzCopy. This flag applies to both files and folders, unless a file-only filter is specified (e.g. **include-pattern** ).

**--preserve-smb-info** string False 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 (e.g. **include-pattern**). The information transferred for folders is the same as that for files, except for Last Write Time which is never preserved for folders.

**--preserve-owner** Only has an effect in when downloading data, and only when the

`--preserve-smb-permissions` is used. If true (the default), the file Owner and Group are preserved in downloads. If this flag is set to false, `--preserve-smb-permissions` will still preserve ACLs but Owner and Group will be based on the user that is running AzCopy.

`--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 sub-directories recursively when uploading from local file system.

`--s2s-detect-source-changed` Check if source has changed after enumerating.

`--s2s-handle-invalid-metadata` string Specifies how invalid metadata keys are handled. Available options: ExcludelfInvalid, FaillfInvalid, RenamelfInvalid. (default "ExcludelfInvalid")

`--s2s-preserve-access-tier` Preserve access tier during service to service copy. Please 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 is not supported, please use `s2sPreserveAccessTier=false` to bypass copying access tier. (default true)

`--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 additional request per object or file. (default true)

## Options inherited from parent commands

`--cap-mbps` uint32 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")

## See also

- [azcopy](#)

# azcopy doc

11/13/2019 • 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]
```

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Options

OPTION	DESCRIPTION
-h, --help	Shows help content for the doc command.

## Options inherited from parent commands

OPTION	DESCRIPTION
--cap-mbps uint32	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".

## See also

- [azcopy](#)

# azcopy env

11/13/2019 • 2 minutes to read • [Edit Online](#)

Shows the environment variables that can configure AzCopy's behavior.

## Synopsis

```
azcopy env [flags]
```

### 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.

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Options

OPTION	DESCRIPTION
-h, --help	Shows help content for the env command.
--show-sensitive	Shows sensitive/secret environment variables.

## Options inherited from parent commands

OPTION	DESCRIPTION
--cap-mbps uint32	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".

## See also

- [azcopy](#)

# azcopy jobs

11/13/2019 • 2 minutes to read • [Edit Online](#)

Sub-commands 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](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Examples

```
azcopy jobs show [jobID]
```

## Options

OPTION	DESCRIPTION
-h, --help	Show help content for the jobs command.

## Options inherited from parent commands

OPTION	DESCRIPTION
--cap-mbps uint32	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".

## See also

- [azcopy](#)
- [azcopy jobs list](#)
- [azcopy jobs resume](#)
- [azcopy jobs show](#)

# azcopy jobs clean

11/13/2019 • 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](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## 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: Canceled, Completed, Failed, InProgress, All (default "All")

## Options inherited from parent commands

**--cap-mbps** uint32 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")

## See also

- [azcopy jobs](#)

# azcopy jobs list

11/13/2019 • 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](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Options

OPTION	DESCRIPTION
-h, --help	Show help content for the list command.

## Options inherited from parent commands

OPTION	DESCRIPTION
--cap-mbps uint32	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".

## See also

- [azcopy jobs](#)

# azcopy jobs remove

11/13/2019 • 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](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Examples

```
azcopy jobs rm e52247de-0323-b14d-4cc8-76e0be2e2d44
```

## Options

**-h, --help** Help for remove.

## Options inherited from parent commands

**--cap-mbps uint32** 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")

## See also

- [azcopy jobs](#)

# azcopy jobs resume

11/13/2019 • 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](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Options

OPTION	DESCRIPTION
--destination-sas string	Destination SAS of the destination for given JobId.
--exclude string	Filter: Exclude these failed transfer(s) when resuming the job. Files should be separated by ':'.
-h, --help	Show help content for the resume command.
--include string	Filter: only include these failed transfer(s) when resuming the job. Files should be separated by ':'.
--source-sas string	source SAS of the source for given JobId.

## Options inherited from parent commands

OPTION	DESCRIPTION
--cap-mbps uint32	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".

## See also

- [azcopy jobs](#)

# azcopy jobs show

11/13/2019 • 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](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Options

OPTION	DESCRIPTION
<code>-h, --help</code>	Shows help content for the show command.
<code>--with-status string</code>	Only list the transfers of job with this status, available values: Started, Success, Failed

## Options inherited from parent commands

OPTION	DESCRIPTION
<code>--cap-mbps uint32</code>	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.
<code>--output-type string</code>	Format of the command's output. The choices include: text, json. The default value is "text".

## See also

- [azcopy jobs](#)

# azcopy list

11/13/2019 • 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](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Examples

```
azcopy list [containerURL]
```

## Options

OPTION	DESCRIPTION
-h, --help	Show help content for the list command.
--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.

## Options inherited from parent commands

OPTION	DESCRIPTION
--cap-mbps uint32	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".

## See also

- [azcopy](#)

# azcopy login

3/26/2020 • 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.

Please refer to the examples for more information.

### 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](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## 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/userAssignedIdentitie
s/myID"
```

Log in as a service principal 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
```

Log in as a service principal using a certificate and password. Set the environment variable AZCOPY\_SPA\_CERT\_PASSWORD to the certificate's password for cert-based service principal authorization.

```
azcopy login --service-principal --certificate-path /path/to/my/cert
```

Make sure to treat /path/to/my/cert as a path to a PEM or PKCS12 file. AzCopy does not reach into the system cert store to obtain your certificate.

--certificate-path is mandatory when doing cert-based service principal auth.

## Options

OPTION	DESCRIPTION
--aad-endpoint	The Azure Active Directory endpoint to use. The default ( <a href="https://login.microsoftonline.com">https://login.microsoftonline.com</a> ) is correct for the public Azure cloud. Set this parameter when authenticating in a national cloud. See <a href="#">Azure AD authentication endpoints</a> .
This flag is 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	Show help content for the login command.
--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.

OPTION	DESCRIPTION
--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 SPN (Service Principal Name) by using a certificate or a secret. The client secret or certificate password must be placed in the appropriate environment variable. Type <code>AzCopy env</code> 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

OPTION	DESCRIPTION
--cap-mbps uint32	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".

## See also

- [azcopy](#)

# azcopy logout

11/13/2019 • 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](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Options

OPTION	DESCRIPTION
-h, --help	Show help content for the logout command.

## Options inherited from parent commands

OPTION	DESCRIPTION
--cap-mbps uint32	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".

## See also

- [azcopy](#)

# azcopy make

11/13/2019 • 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](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Examples

```
azcopy make "https://[account-name].[blob,file,dfs].core.windows.net/[top-level-resource-name]"
```

## Options

OPTION	DESCRIPTION
-h, --help	Show help content for the make command.
--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

OPTION	DESCRIPTION
--cap-mbps uint32	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".

## See also

- [azcopy](#)

# azcopy remove

11/13/2019 • 2 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](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Examples

Remove a single blob with SAS:

```
azcopy rm "https://[account].blob.core.windows.net/[container]/[path/to/blob]?[SAS]"
```

Remove an entire virtual directory with a SAS:

```
azcopy rm "https://[account].blob.core.windows.net/[container]/[path/to/directory]?[SAS]" --recursive=true
```

Remove only the top blobs inside a virtual directory but not its sub-directories:

```
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: 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="*.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="foo*;*bar"
```

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
file content:
dir1/dir2
blob1
blob2
```

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 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

**--exclude-path** string Exclude these paths when removing. This option does not 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

**-h, --help** help for remove

**--include-path** string Include only these paths when removing. This option does not 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.

**--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")

**--recursive** Look into sub-directories recursively when syncing between directories.

## Options inherited from parent commands

OPTION	DESCRIPTION
--cap-mbps uint32	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".

## See also

- [azcopy](#)

# azcopy sync

4/22/2020 • 4 minutes to read • [Edit Online](#)

Replicates the source location to the destination location.

## 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 (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)

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 'deleteDestination' flag is set to true or prompt, then sync will delete files and blobs at the destination that are not present at the source.

## Related conceptual articles

- [Get started with AzCopy](#)
- [Transfer data with AzCopy and Blob storage](#)
- [Transfer data with AzCopy and file storage](#)
- [Configure, optimize, and troubleshoot AzCopy](#)

## Advanced

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 (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.

```
azcopy sync <source> <destination> [flags]
```

# Examples

Sync a single file:

```
azcopy sync "/path/to/file.txt" "https://[account].blob.core.windows.net/[container]/[path/to/blob]"
```

## NOTE

The destination blob *must* exist. Use `azcopy copy` to copy a single file that does not yet exist in the destination.

Otherwise, the following error occurs:

```
Cannot perform sync due to error: sync must happen between source and destination of the same type, e.g.
either file <-> file, or directory/container <-> directory/container
```

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 sync "/path/to/file.txt" "https://[account].blob.core.windows.net/[container]/[path/to/blob]" --put-md5
```

Sync an entire directory including its sub-directories (note that recursive is on by default):

```
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 top files inside a directory but not its sub-directories:

```
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="*.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="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/exclude flags are used together, only files matching the include patterns would be looked at, but those matching the exclude patterns would be always be 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")

**--delete-destination** string Defines whether to delete extra files from the destination that are not 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")

**--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 copying. This option does not support wildcard characters (\*). Checks relative path prefix(For example: myFolder;myFolder/subDirName/file.pdf). When used in combination with account traversal, paths do not include the container name.

**--exclude-pattern** string Exclude files where the name matches the pattern list. For example:  
\*.jpg;\*.pdf;exactName

**-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

**--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")

--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 sub-directories recursively when syncing between directories. (default true). (default true)

## Options inherited from parent commands

OPTION	DESCRIPTION
--cap-mbps uint32	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".

## See also

- [azcopy](#)

# Known issues with Azure Data Lake Storage Gen2

4/21/2020 • 3 minutes to read • [Edit Online](#)

This article describes limitations and known issues of Azure Data Lake Storage Gen2.

## 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](#).

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 [Open source platforms that support Azure Data Lake Storage Gen2](#).

## Blob storage APIs

Blob APIs and Data Lake Storage Gen2 APIs can operate on the same data.

This section describes issues and limitations with using blob APIs and Data Lake Storage Gen2 APIs to operate on the same data.

- You cannot use both Blob APIs 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 APIs, then that file's blocks won't be visible to calls to the [Get Block List](#) blob API. You can overwrite a file by using either Data Lake Storage Gen2 APIs or Blob APIs. This won't affect file properties.
- 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](#)
- [Put Blob \(Append\)](#)

- [Append Block](#)
- [Append Block 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.

## File system support in SDKs, PowerShell, and Azure CLI

- Get and set ACL operations are not currently recursive.
- [Azure CLI](#) support is in public preview.

## Lifecycle management policies

- The deletion of blob snapshots is not yet supported.

## Archive Tier

There is currently a bug that affects the archive access tier.

## Blobfuse

Blobfuse is not supported.

## 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.

## 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.

## Windows Azure Storage Blob (WASB) driver (unsupported with Data Lake Storage Gen2)

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.

For the time being (and most likely the foreseeable future), we won't support customers using the WASB driver as a client to a hierarchical namespace-enabled storage account. Instead, we recommend that you opt to use the [Azure Blob File System \(ABFS\)](#) driver in your Hadoop environment. If you are trying to migrate off of an on-premise 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.

# Microsoft client tools for working with Azure Storage

9/29/2019 • 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
Azure portal	Web	Yes	Yes	Yes	Yes	Yes	Yes
Azure Storage Explorer	Windows, OSX	Yes	Yes	Yes	Yes	Yes	Yes
Microsoft Visual Studio Cloud Explorer	Windows	Yes	Yes	Yes	Yes	Yes	No

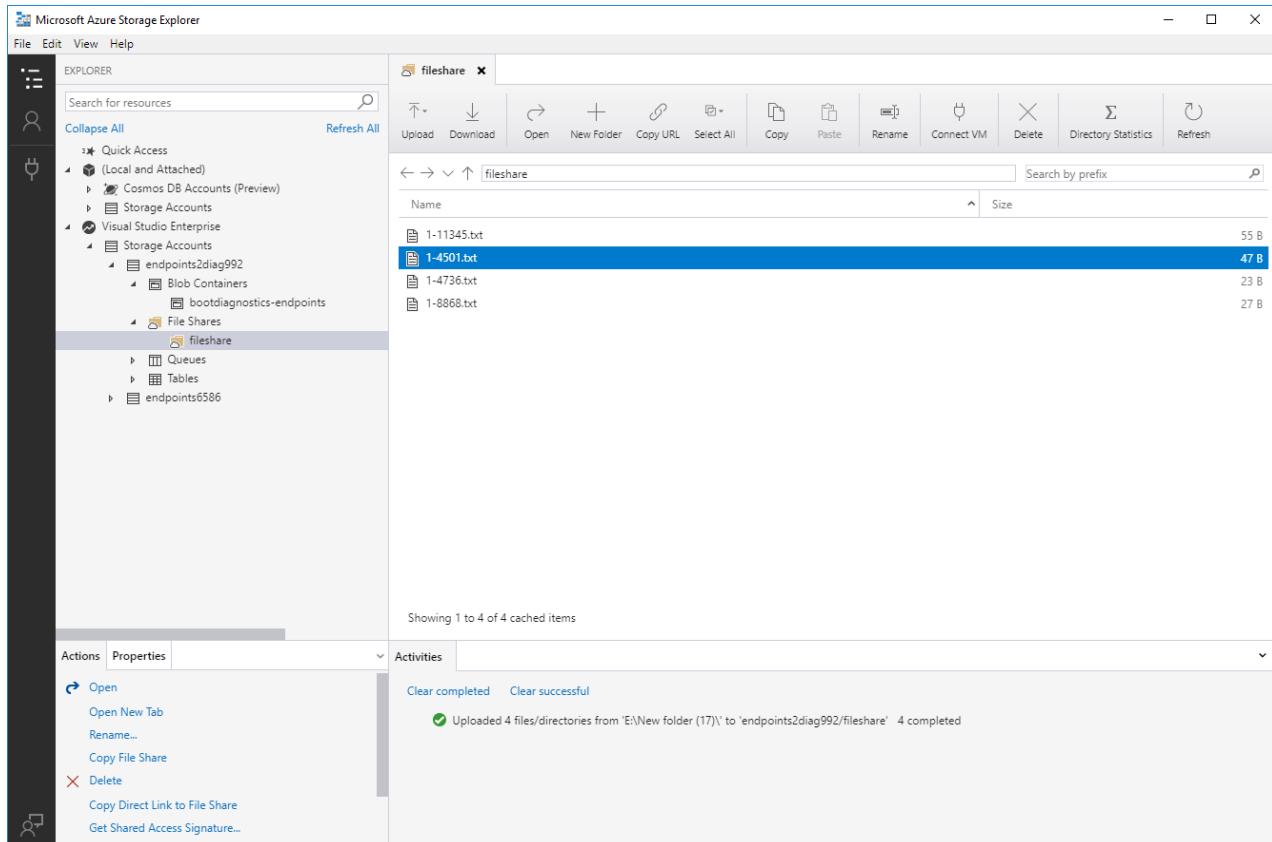
There are also a number of third-party tools available for working with Azure Storage data.

# Get started with Storage Explorer

11/8/2019 • 10 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 10 (recommended)
- Windows 8
- Windows 7

For all versions of Windows, Storage Explorer requires .NET Framework 4.6.2 or later.

## 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 storage accounts. In general you can either:

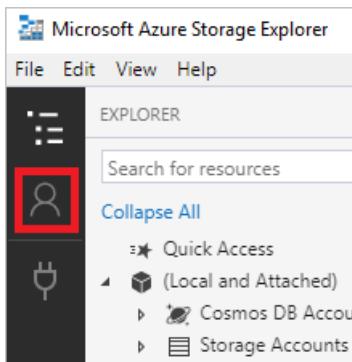
- [Sign in to Azure to access your subscriptions and their resources](#)
- [Attach a specific Storage or CosmosDB resource](#)

## Sign in to Azure

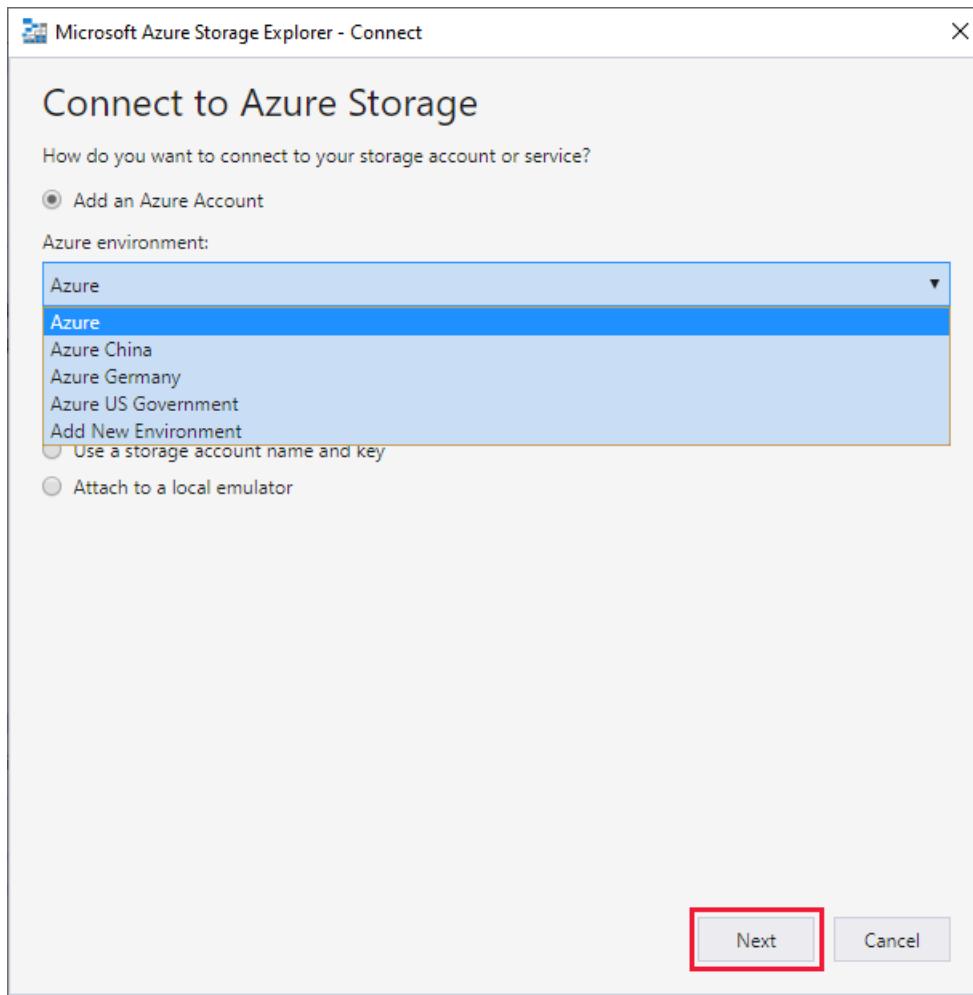
### NOTE

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, which give you access to your storage account, the containers in the account, and the data in the containers. If you have permissions only at the data layer, consider [adding a resource through Azure AD](#). 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.

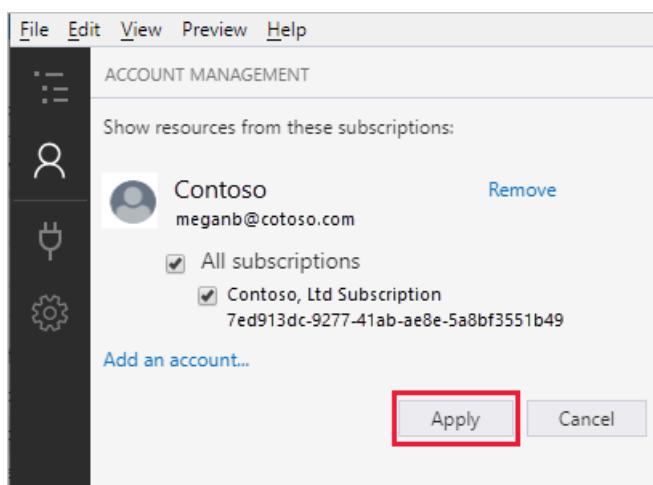


2. **ACCOUNT MANAGEMENT** now displays all the Azure accounts you've signed in to. To connect to another account, select **Add an account**.
3. In **Connect to Azure Storage**, select an Azure cloud from **Azure environment** to sign in to a national cloud or an Azure Stack. After you choose your environment, select **Next**.

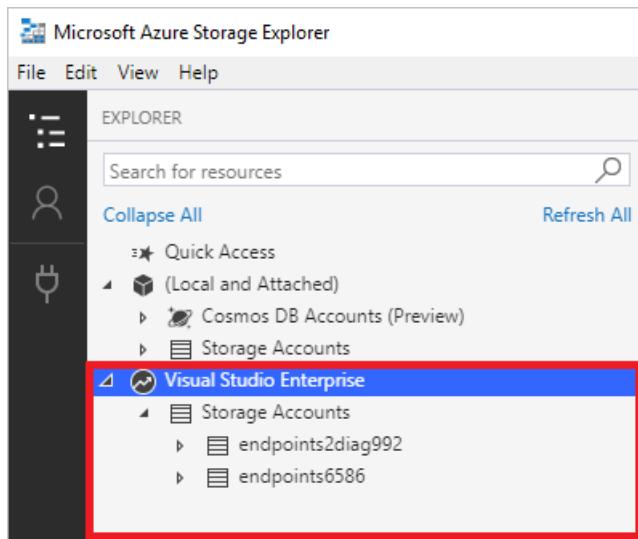


Storage Explorer opens a page for you to sign in. For more information, see [Connect storage explorer to an Azure Stack subscription or storage account](#).

4. After you successfully sign in with an Azure account, the account and the Azure subscriptions associated with that account appear under **ACCOUNT MANAGEMENT**. Select **All subscriptions** to toggle your selection between all or none of the listed Azure subscriptions. Select the Azure subscriptions that you want to work with, and then select **Apply**.



EXPLORER displays the storage accounts associated with the selected Azure subscriptions.



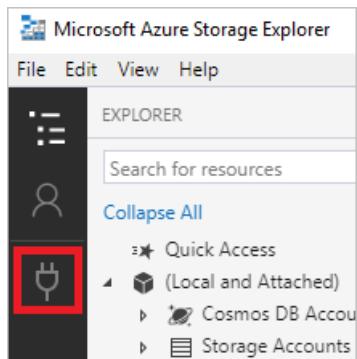
## Attach a specific resource

There are several ways to attach to a resource in Storage Explorer:

- [Add a resource via Azure AD](#). If you have permissions only at the data layer, use this option to add a blob container or an Azure Data Lake Storage Gen2 Blob storage container.
- [Use a connection string](#). Use this option if you have a connection string to a storage account. Storage Explorer supports both key and [shared access signature](#) connection strings.
- [Use a shared access signature URI](#). If you have a [shared access signature URI](#) to a blob container, file share, queue, or table, use it to attach to the resource. To get a shared access signature URI, you can either use [Storage Explorer](#) or the [Azure portal](#).
- [Use a name and key](#). If you know either of the account keys to your storage account, you can use this option to quickly connect. Find your keys in the storage account page by selecting **Settings > Access keys** in the [Azure portal](#).
- [Attach to a local emulator](#). If you're using one of the available Azure Storage emulators, use this option to easily connect to your emulator.
- [Connect to an Azure Cosmos DB account by using a connection string](#). Use this option if you have a connection string to a CosmosDB instance.
- [Connect to Azure Data Lake Store by URI](#). Use this option if you have a URI to Azure Data Lake Store.

### Add a resource via Azure AD

1. Select the **Connect** symbol to open **Connect to Azure Storage**.



2. If you haven't already done so, use the **Add an Azure Account** option to sign in to the Azure account that has access to the resource. After you sign in, return to **Connect to Azure Storage**.
3. Select **Add a resource via Azure Active Directory (Azure AD)**, and then select **Next**.
4. Select an Azure account and tenant. These values must have access to the Storage resource you want to attach to. Select **Next**.

5. Choose the resource type you want to attach. Enter the information needed to connect.

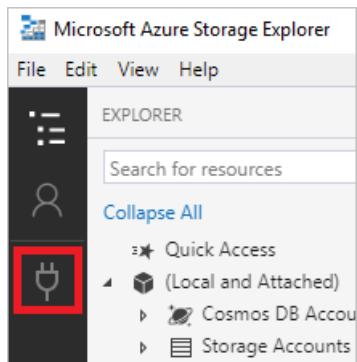
The information you enter on this page depends on what type of resource you're adding. Make sure to choose the correct type of resource. After you've entered the required information, select **Next**.

6. Review the **Connection Summary** to make sure all the information is correct. If it is, select **Connect**. Otherwise, select **Back** to return to the previous pages to fix any incorrect information.

After the connection is successfully added, the resource tree goes to the node that represents the connection. The resource appears under **Local & Attached > Storage Accounts > (Attached Containers) > 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](#).

#### Use a connection string

1. Select the **Connect** symbol to open **Connect to Azure Storage**.



2. Select **Use a connection string**, and then select **Next**.

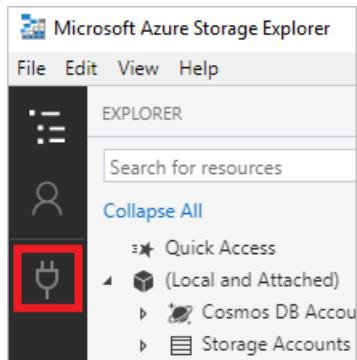
3. Choose a display name for your connection and enter your connection string. Then, select **Next**.

4. Review the **Connection Summary** to make sure all the information is correct. If it is, select **Connect**. Otherwise, select **Back** to return to the previous pages to fix any incorrect information.

After the connection is successfully added, the resource tree goes to the node that represents the connection. The resource appears under **Local & Attached > Storage Accounts**. 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](#).

#### Use a shared access signature URI

1. Select the **Connect** symbol to open **Connect to Azure Storage**.



2. Select **Use a shared access signature (SAS) URI**, and then select **Next**.

3. Choose a display name for your connection and enter your shared access signature URI. The service endpoint for the type of resource you're attaching should autofill. If you're using a custom endpoint, it's possible it might not. Select **Next**.

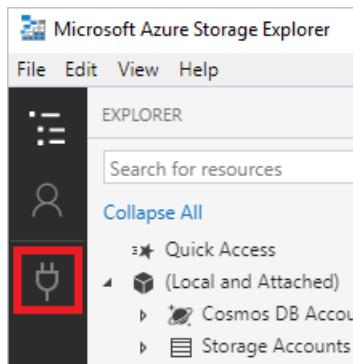
4. Review the **Connection Summary** to make sure all the information is correct. If it is, select **Connect**.

Otherwise, select **Back** to return to the previous pages to fix any incorrect information.

After the connection is successfully added, the resource tree goes to the node that represents the connection. The resource appears under **Local & Attached > Storage Accounts > (Attached Containers) > the service node for the type of container you attached**. If Storage Explorer couldn't add your connection, see the [Azure Storage Explorer troubleshooting guide](#). See the troubleshooting guide if you can't access your data after successfully adding the connection.

#### Use a name and key

1. Select the **Connect** symbol to open **Connect to Azure Storage**.



2. Select **Use a storage account name and key**, and then select **Next**.
3. Choose a display name for your connection.
4. Enter your storage account name and either of its access keys.
5. Choose the **Storage domain** to use and then select **Next**.
6. Review the **Connection Summary** to make sure all the information is correct. If it is, select **Connect**. Otherwise, select **Back** to return to the previous pages to fix any incorrect information.

After the connection is successfully added, the resource tree goes to the node that represents the connection. The resource appears under **Local & Attached > Storage Accounts**. 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](#).

#### Attach to a local emulator

Storage Explorer currently supports two official Storage emulators:

- [Azure Storage emulator](#) (Windows only)
- [Azurite](#) (Windows, macOS, or Linux)

If your emulator is listening on the default ports, you can use the **Emulator - Default Ports** node to access your emulator. Look for **Emulator - Default Ports** under **Local & Attached > Storage Accounts**.

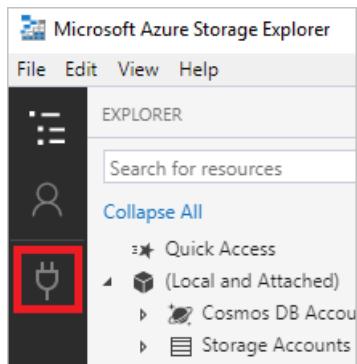
If you want to use a different name for your connection, or if your emulator isn't running on the default ports, follow these steps:

1. Start your emulator. Enter the command `AzureStorageEmulator.exe status` to display the ports for each service type.

#### IMPORTANT

Storage Explorer doesn't automatically start your emulator. You must start it manually.

2. Select the **Connect** symbol to open **Connect to Azure Storage**.



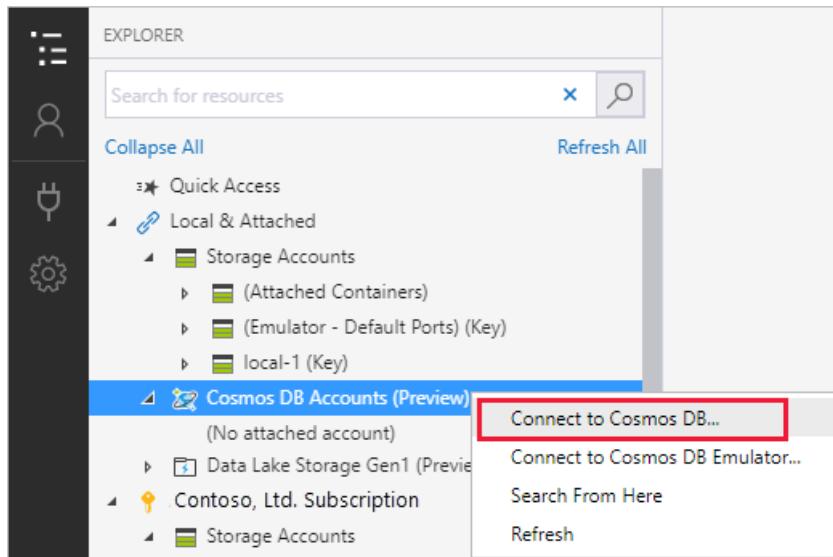
3. Select **Attach to a local emulator**, and then select **Next**.
4. Choose a display name for your connection and enter the ports your emulator is listening on for each service type. **Attach to a Local Emulator** suggests the default port values for most emulators. **Files port** is blank, because neither of the official emulators currently support the Files service. If the emulator you're using does support Files, you can enter the port to use. Then, select **Next**.
5. Review the **Connection Summary** and make sure all the information is correct. If it is, select **Connect**. Otherwise, select **Back** to return to the previous pages to fix any incorrect information.

After the connection is successfully added, the resource tree goes to the node that represents the connection. The node should appear under **Local & Attached > Storage Accounts**. 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](#).

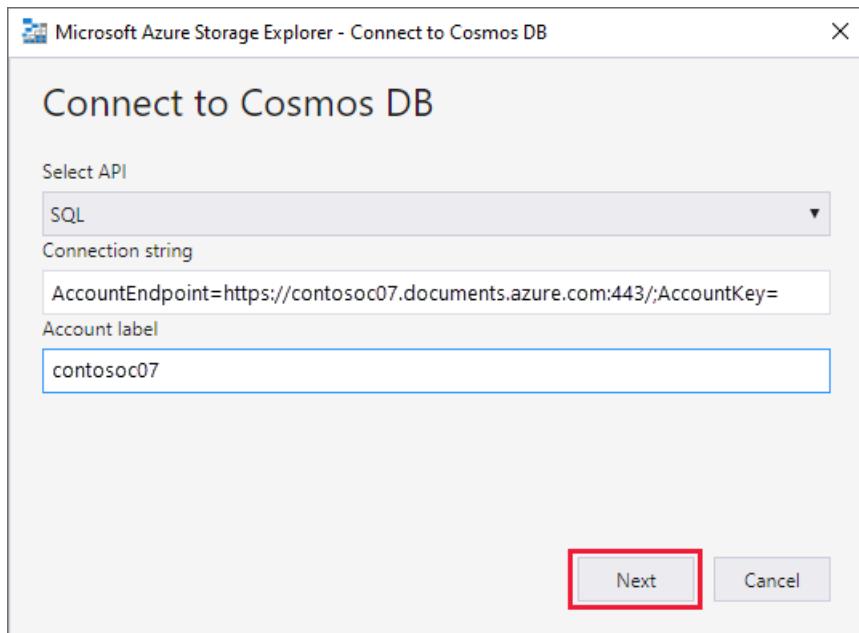
#### Connect to an Azure Cosmos DB account by using a connection string

Instead of managing Azure Cosmos DB accounts through an Azure subscription, you can connect to Azure Cosmos DB by using a connection string. To connect, follow these steps:

1. Under EXPLORER, expand **Local & Attached**, right-click **Cosmos DB Accounts**, and select **Connect to Azure Cosmos DB**.



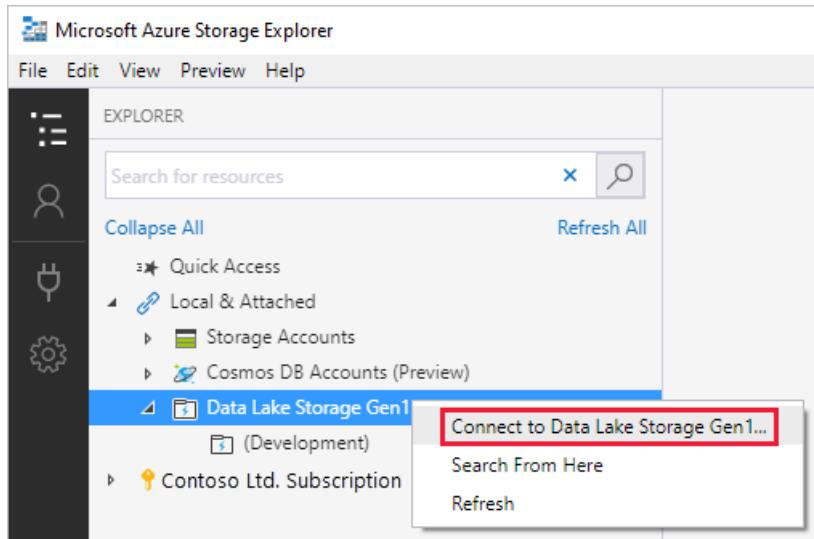
2. Select the Azure Cosmos DB API, enter your **Connection String** data, and then select **OK** to connect the Azure Cosmos DB account. For information about how to retrieve the connection string, see [Manage an Azure Cosmos account](#).



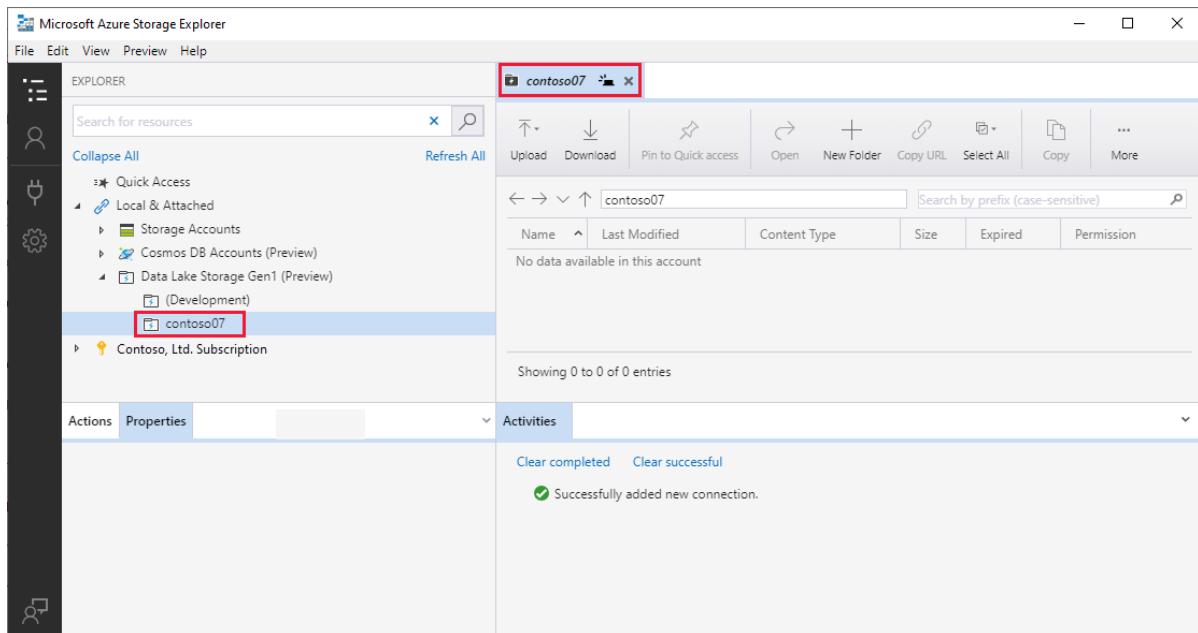
#### Connect to Azure Data Lake Store by URI

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.

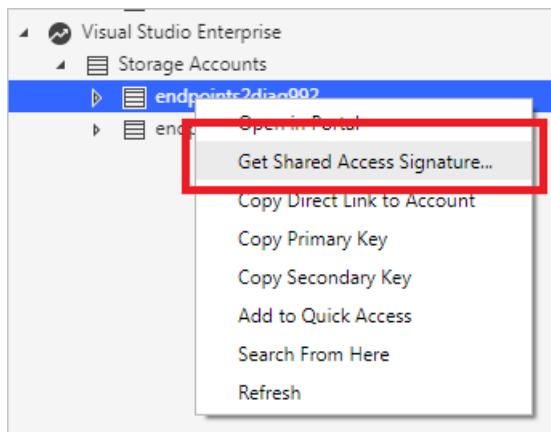


This example uses Data Lake Storage Gen1. Azure Data Lake Storage Gen2 is now available. For more information, see [What is Azure Data Lake Storage Gen1](#).

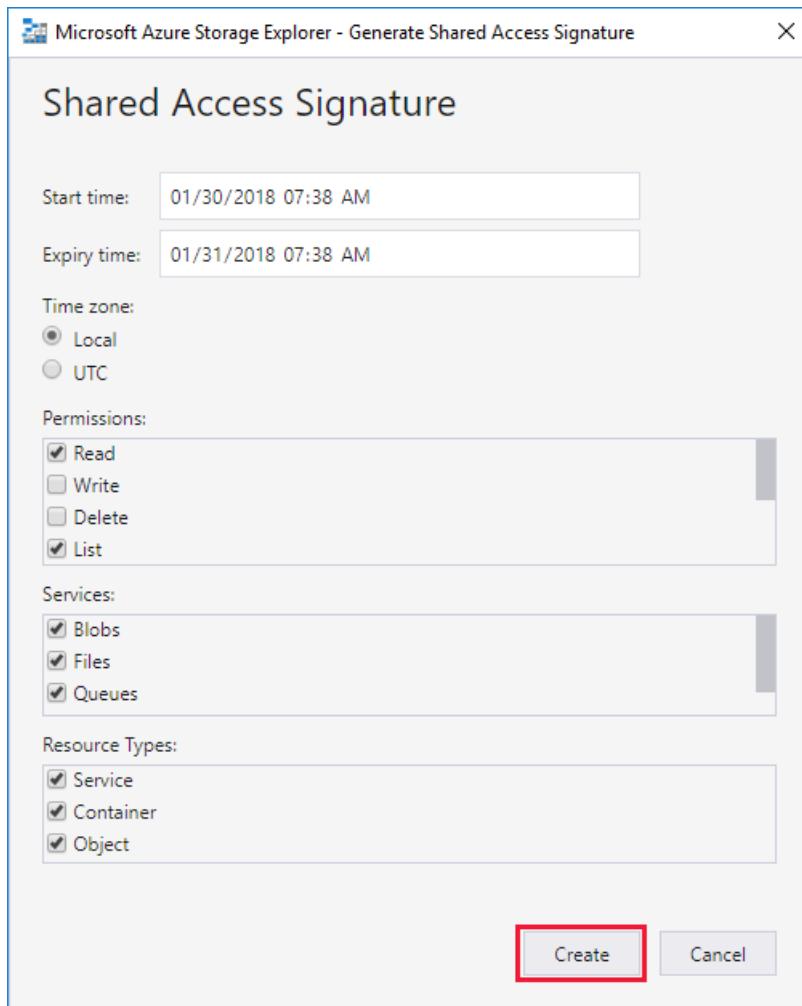
## Generate a shared access signature in Storage Explorer

### 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.

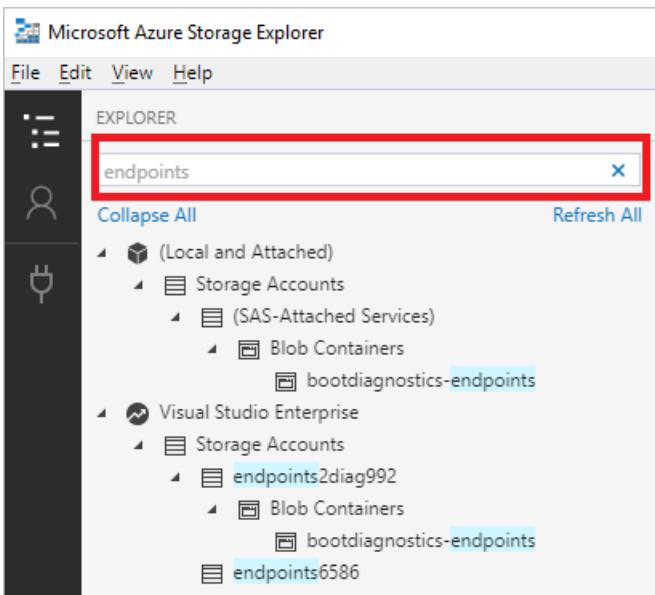
#### Service level shared access signature

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](#)
- [Work with data using Azure Storage Explorer](#)
- [Manage Azure Data Lake Store resources with Storage Explorer](#)

# Azure Storage Explorer troubleshooting guide

3/31/2020 • 17 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.

## RBAC permissions issues

Role-based access control [RBAC](#) enables highly granular access management of Azure resources by combining sets of permissions into *roles*. Here are some strategies to get RBAC working optimally in Storage Explorer.

### How do I access my resources in Storage Explorer?

If you're having problems accessing storage resources through 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 administrator 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. Therefore, 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.

RBAC roles can contain 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 administrator?

We don't currently have an RBAC-related solution for this issue. As a workaround, you can request a SAS URI to [attach to your resource](#).

### Recommended built-in RBAC roles

There are several built-in RBAC roles which can provide the permissions needed to use Storage Explorer. Some of those roles are:

- [Owner](#): Manage everything, including access to resources. **Note**: this role will give you key access.
- [Contributor](#): Manage everything, excluding access to resources. **Note**: this role will give you key access.
- [Reader](#): Read and list resources.
- [Storage Account Contributor](#): Full management of storage accounts. **Note**: this role will give you key access.
- [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.

## Error: Self-signed certificate in certificate chain (and similar errors)

Certificate errors typically occur in one of the following situations:

- The app is connected through a *transparent proxy*, which means a server (such as your company server) is intercepting HTTPS traffic, decrypting it, and then encrypting it by using a self-signed certificate.
- You're running an application that's injecting a self-signed TLS/SSL certificate into the HTTPS messages that you receive. Examples of applications that inject certificates include antivirus and network traffic inspection software.

When Storage Explorer sees a self-signed or untrusted certificate, it no longer knows whether the received HTTPS message has been altered. If you have a copy of the self-signed certificate, you can instruct Storage Explorer to trust it by following these steps:

1. Obtain a Base-64 encoded X.509 (.cer) copy of the certificate.
2. Go to **Edit > SSL Certificates > Import Certificates**, and then use the file picker to find, select, and open the .cer file.

This issue may also occur if there are multiple certificates (root and intermediate). To fix this error, both certificates must be added.

If you're unsure of where the certificate is coming from, follow these steps to find it:

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 `s_client -showcerts -connect microsoft.com:443`.
4. Look for self-signed certificates. If you're unsure of which certificates are self-signed, make note of anywhere the subject `("s:")` and issuer `("i:")` are the same.
5. When you find self-signed certificates, for each one, copy and paste everything from (and including)

-----BEGIN CERTIFICATE----- through -----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 that you created.

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 by using the `--ignore-certificate-errors` flag. When opened with this flag, Storage Explorer ignores certificate errors.

## Sign-in issues

### Blank sign-in dialog box

Blank sign-in dialog boxes most often occur when Active Directory Federation Services (AD FS) prompts Storage Explorer to perform a redirect, which is unsupported by Electron. To work around this issue, you can try to use Device Code Flow for sign-in. To do so, follow these steps:

1. On the left vertical tool bar, open **Settings**. In the Settings Panel, go to **Application > Sign in**. Enable **Use device code flow sign-in**.
2. Open the **Connect** dialog box (either through the plug icon on the left-side vertical bar or by selecting **Add Account** on the account panel).
3. Choose the environment you want to sign in to.
4. Select **Sign In**.
5. Follow the instructions on the next panel.

If you can't sign in to the account you want to use because your default browser is already signed in to a different account, do one of the following:

- Manually copy the link and code into a private session of your browser.
- Manually copy the link and code into a different browser.

### Reauthentication loop or UPN change

If you're in a reauthentication loop or have changed the UPN of one of your accounts, follow these steps:

1. Remove all accounts and then close Storage Explorer.
2. 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.
3. If you're running Mac or Linux, you'll 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.

### Conditional Access

Because of a limitation in the Azure AD Library used by Storage Explorer, Conditional Access isn't supported when Storage Explorer is being used on Windows 10, Linux, or macOS.

## Mac Keychain errors

The macOS Keychain can sometimes enter a state that causes issues for the Storage Explorer authentication library. To get the Keychain out of this state, follow these steps:

1. Close Storage Explorer.
2. Open Keychain (press Command+Spacebar, type **keychain**, and press Enter).
3. Select the "login" Keychain.
4. Select the padlock icon to lock the Keychain. (The padlock will appear locked when the process is complete. 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 **Allow** if **Always Allow** isn't available).
7. Try to sign in.

#### General sign-in troubleshooting steps

- If you're on macOS, and the sign-in window never appears over the **Waiting for authentication** dialog box, try [these steps](#).
- Restart Storage Explorer.
- If the authentication window is blank, wait at least one minute before closing the authentication dialog box.
- Make sure that your proxy and certificate settings are properly configured for both your machine and Storage Explorer.
- If you're running Windows and have access to Visual Studio 2019 on the same machine and to the sign-in credentials, try signing in to Visual Studio 2019. After a successful sign-in to Visual Studio 2019, you can open Storage Explorer and see your account in the account panel.

If none of these methods work, [open an issue in 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 (Azure, Azure China 21Vianet, Azure Germany, Azure US Government, or Custom Environment).
- If you're behind a proxy server, make sure you've configured the Storage Explorer proxy correctly.
- Try removing and re-adding the account.
- If there's a "More information" 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, feel free to [open an issue in GitHub](#).

## Can't remove an attached account or storage 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`

#### NOTE

Close Storage Explorer before you delete these folders.

#### NOTE

If you have 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

First, make sure that the following information you entered is correct:

- The proxy URL and port number
- Username and password if the proxy requires them

#### NOTE

Storage Explorer doesn't support proxy auto-config files for configuring proxy settings.

## Common solutions

If you're still experiencing issues, try the following troubleshooting methods:

- If you can connect to the internet without using your proxy, verify that Storage Explorer works without proxy settings enabled. If this is the case, there may be an issue with your proxy settings. Work with your administrator to identify the problems.
- Verify that other applications that use the proxy server work as expected.
- Verify that you can connect to the portal for the Azure environment you're trying to use.
- 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 `InvalidQueryParameterValue` or a similar XML response.
- If someone else is also using Storage Explorer with your proxy server, verify that they can connect. If they can, you may have to contact your proxy server admin.

## Tools for diagnosing issues

If you have networking tools, such as Fiddler for Windows, you can diagnose the problems as follows:

- If you have to work through your proxy, you may have to configure your networking tool to connect through the proxy.
- Check the port number used by your networking tool.
- Enter the local host URL and the networking tool's port number as proxy settings in Storage Explorer. When you do this correctly, your networking tool starts logging network requests made by Storage Explorer to management and service endpoints. For example, enter `https://cawablobgrs.blob.core.windows.net/` for your blob endpoint in a browser, and you'll receive a response that resembles the following:

```
<?xml version="1.0" encoding="UTF-8"?>
- <Error>
 <Code>InvalidQueryParameterValue</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>
```

This response suggests the resource exists, even though you can't access it.

## Contact proxy server admin

If your proxy settings are correct, you may 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 doesn't currently 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 you're granted access to a resource from the owner of the subscription or account, 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. You can do this by right-clicking the node for your storage account and selecting **Open in Portal**. Then, go to the **Access Keys** blade. 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 SAS 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** (left side) > **file://**.
3. Depending on the type of connection you're having an issue with, look for its key and then copy its value into a text editor. The value is an array of your custom connection names, like the following:
  - 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 `[]`.

If you want to preserve the connections that aren't corrupted, you can use the following steps to locate the corrupted connections. If you don't mind losing all existing connections, you can 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, and 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 have examined all your connections.

After going through all your connections, for all connections names that aren't added back, you must clear their corrupted data (if there is any) and 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 (for example, `StorageExplorer_CustomConnections_Accounts_v1/account1` ).
4. Delete these entries and re-add the connections.

If you still encounter this error after running these steps, or if you want to share what you suspect has corrupted the connections, [open an issue](#) on our GitHub page.

## Issues with SAS URL

If you're connecting to a service through a SAS URL and experiencing an error:

- Verify that the URL provides the necessary permissions to read or list resources.
- Verify that the URL has not expired.
- If the SAS URL is based on an access policy, verify that the access policy has not been revoked.

If you accidentally attached by using an invalid SAS URL and now cannot detach, follow these steps:

1. When you're running Storage Explorer, press F12 to open the Developer Tools window.
2. On the **Application** tab, select **Local Storage > file://** in the tree on the left.
3. Find the key associated with the service type of the problematic SAS URI. For example, if the bad SAS 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 then delete it.
5. Press Ctrl+R to reload Storage Explorer.

## Linux dependencies

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, and 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
```

You can also download the application as a .tar.gz file, but you'll have to install dependencies manually.

## IMPORTANT

Storage Explorer as provided in the .tar.gz download is supported only for Ubuntu distributions. Other distributions haven't been verified and may require alternative or additional packages.

These packages are the most common requirements for Storage Explorer on Linux:

- [.NET Core 2.2 Runtime](#)
- `libgconf-2-4`
- `libgnome-keyring0` or `libgnome-keyring-dev`
- `libgnome-keyring-common`

## NOTE

Storage Explorer version 1.7.0 and earlier require .NET Core 2.0. If you have a newer version of .NET Core installed, you'll have to [patch Storage Explorer](#). If you're running Storage Explorer 1.8.0 or later, you should be able to use up to .NET Core 2.2.

Versions beyond 2.2 have not been verified to work at this time.

- [Ubuntu 19.04](#)
- [Ubuntu 18.04](#)
- [Ubuntu 16.04](#)
- [Ubuntu 14.04](#)

1. Download Storage Explorer.
2. Install the [.NET Core Runtime](#).
3. Run the following command:

```
sudo apt-get install libgconf-2-4 libgnome-keyring0
```

## Patching 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](#). 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" from the Azure portal doesn't work

If the **Open In Explorer** button on the Azure portal doesn't work, make sure you're using a compatible browser.

The following browsers have been tested for compatibility:

- Microsoft Edge
- Mozilla Firefox
- Google Chrome
- Microsoft Internet Explorer

## Next steps

If none of these solutions work for you, [open an issue in GitHub](#). You can also do this by selecting the **Report issue to GitHub** button in the lower-left corner.

Actions Properties	
URL	<a href="https://">https://</a>
Type	Blob C
Public Read Access	Off
Lease State	availab
Lease Status	unlocke
Last Modified	Mon, 2

# Storage Explorer Accessibility

4/17/2019 • 2 minutes to read • [Edit Online](#)

## Screen Readers

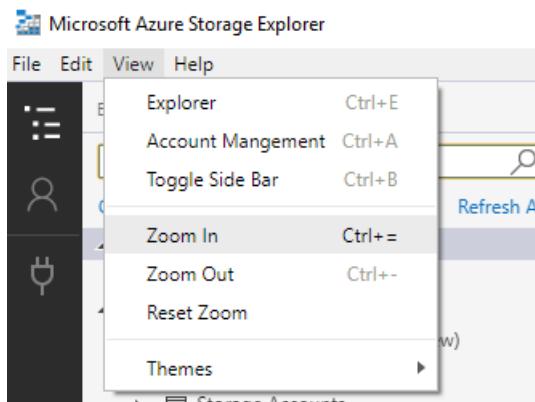
Storage Explorer supports the use of a screen reader on Windows and Mac. The following screen readers are recommended for each platform:

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

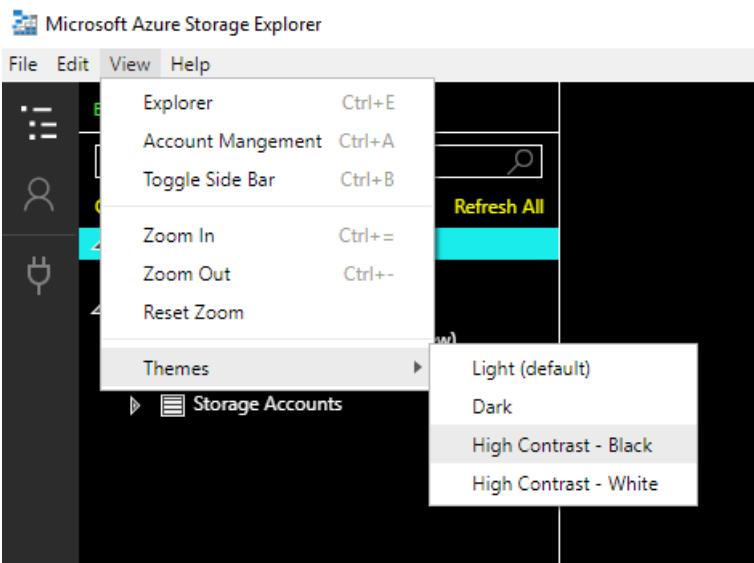
You can make the text in Storage Explorer larger via zooming in. To zoom in, click on **Zoom In** in the Help menu. You can also use the Help menu to zoom out and reset the zoom level back to the default level.



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 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