

Contents

[Azure Files documentation](#)

[Overview](#)

[What is Azure Files?](#)

[Quickstarts](#)

[Create / use file shares - Windows](#)

[Create file shares - Portal](#)

[Create file shares - PowerShell](#)

[Create file shares - CLI](#)

[Create file shares - Storage Explorer](#)

[Tutorials](#)

[Extend Windows file servers with Azure File Sync](#)

[Concepts](#)

[Planning for an Azure Files deployment](#)

[Planning for an Azure File Sync deployment](#)

[Networking considerations for direct access](#)

[About cloud tiering](#)

[About storage accounts](#)

[Azure file share snapshots](#)

[Scalability and performance targets](#)

[Identity-based authentication and authorization](#)

[Encryption at-rest](#)

[Compliance offerings](#)

[Development](#)

[Manage concurrency](#)

[Shared key authentication](#)

[Shared access signatures \(SAS\)](#)

[Monitor and troubleshoot](#)

[Monitor, diagnose, and troubleshoot](#)

[Metrics in Azure Monitor](#)

[Migrate to new metrics](#)

[Storage analytics metrics \(classic\)](#)

[FAQ](#)

[How-to guides](#)

[Create](#)

[Create a file share](#)

[Enable large file shares](#)

[Create a premium file share](#)

[Deploy](#)

[Deploy Azure Files](#)

[Deploy Azure File Sync](#)

[Configure File Sync proxy and firewall settings](#)

[SQL Server FCI with premium file shares](#)

[Mount](#)

[Use Azure Files with Windows](#)

[Use Azure Files with Linux](#)

[Use Azure Files with macOS](#)

[Network](#)

[Configure Site-to-Site VPN](#)

[Configure Point-to-Site VPN on Windows](#)

[Configure Point-to-Site VPN on Linux](#)

[Manage](#)

[Register a server with Azure File Sync](#)

[Add an Azure File Sync Server endpoint](#)

[Manage storage in Azure independent clouds](#)

[Change how data is replicated](#)

[Initiate an account failover](#)

[Monitor](#)

[Monitor Azure File Sync](#)

[Migrate](#)

[Migrate to Azure File Sync](#)

[Migrate to Azure File Sync from StorSimple](#)

Develop

[Configure connection strings](#)

[.NET](#)

[Java](#)

[C++](#)

[Python](#)

Secure

[Manage encryption keys for the storage account](#)

[Check the encryption key model for the account](#)

[Configure customer-managed encryption keys](#)

[Portal](#)

[PowerShell](#)

[Azure CLI](#)

[Configure firewalls and virtual networks](#)

[Require secure transfer](#)

[Enable AD authentication and authorization](#)

[Enable Azure AD DS authentication and authorization](#)

[Enable secure TLS for Azure Storage client](#)

Troubleshoot

[Troubleshoot Azure Files on Windows](#)

[Troubleshoot Azure Files on Linux](#)

[Troubleshoot Azure Files performance](#)

[Troubleshoot Azure File Sync](#)

[Troubleshoot Deleting Azure Files](#)

[Recover a deleted storage account](#)

Reference

[Azure PowerShell](#)

[Azure CLI](#)

[.NET](#)

[Files \(version 11.x\)](#)

[Data Movement](#)

[Storage Resource Provider](#)

[Java](#)

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

[Storage Resource Provider](#)

[JavaScript \(version 10.x\)](#)

[Python \(version 2.x\)](#)

[REST](#)

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

[Storage Resource Provider](#)

[Import/Export](#)

[C++](#)

[Ruby](#)

[PHP](#)

[iOS](#)

[Android](#)

[Resource Manager template](#)

[Samples](#)

[Resources](#)

[Azure updates](#)

[Azure File Sync release notes](#)

[Azure Storage forum](#)

[Azure Storage on Stack Overflow](#)

[Pricing for Azure storage](#)

[Azure pricing calculator](#)

[Videos](#)

[NuGet packages \(.NET\)](#)

[Microsoft.Azure.Storage.Common \(version 11.x\)](#)

[Azure.Storage.Common \(version 12.x - preview\)](#)

[Microsoft.Azure.Storage.File \(version 11.x\)](#)

[Azure.Storage.File \(version 12.x - preview\)](#)

[Azure Configuration Manager](#)

[Azure Storage Data Movement library](#)

[Storage Resource Provider library](#)

Source code

.NET

[Azure Storage client library](#)

[Version 12.x \(preview\)](#)

[Version 11.x and earlier](#)

[Data Movement library](#)

[Storage Resource Provider library](#)

Java

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

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

Node.js

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

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

Python

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

[Azure Storage client library version 2.1](#)

What is Azure Files?

12/16/2019 • 3 minutes to read • [Edit Online](#)

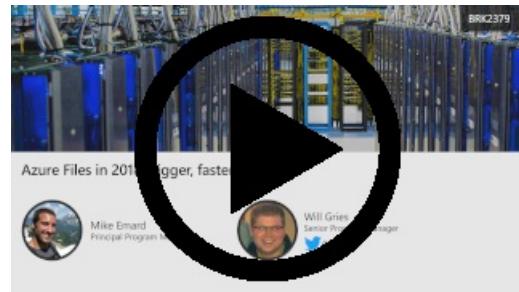
Azure Files offers fully managed file shares in the cloud that are accessible via the industry standard [Server Message Block \(SMB\) protocol](#). Azure file shares can be mounted concurrently by cloud or on-premises deployments of Windows, Linux, and macOS. Additionally, Azure file shares can be cached on Windows Servers with Azure File Sync for fast access near where the data is being used.

Videos

INTRODUCING AZURE FILE SYNC (2 M)



AZURE FILES WITH SYNC (IGNITE 2017) (85 M)



Why Azure Files is useful

Azure file shares can be used to:

- **Replace or supplement on-premises file servers:**

Azure Files can be used to completely replace or supplement traditional on-premises file servers or NAS devices. Popular operating systems such as Windows, macOS, and Linux can directly mount Azure file shares wherever they are in the world. Azure file shares can also be replicated with Azure File Sync to Windows Servers, either on-premises or in the cloud, for performance and distributed caching of the data where it's being used.

- **"Lift and shift" applications:**

Azure Files makes it easy to "lift and shift" applications to the cloud that expect a file share to store file application or user data. Azure Files enables both the "classic" lift and shift scenario, where both the application and its data are moved to Azure, and the "hybrid" lift and shift scenario, where the application data is moved to Azure Files, and the application continues to run on-premises.

- **Simplify cloud development:**

Azure Files can also be used in numerous ways to simplify new cloud development projects. For example:

- **Shared application settings:**

A common pattern for distributed applications is to have configuration files in a centralized location where they can be accessed from many application instances. Application instances can load their configuration through the File REST API, and humans can access them as needed by mounting the SMB share locally.

- **Diagnostic share:**

An Azure file share is a convenient place for cloud applications to write their logs, metrics, and crash dumps. Logs can be written by the application instances via the File REST API, and developers can access them by mounting the file share on their local machine. This enables great

flexibility, as developers can embrace cloud development without having to abandon any existing tooling they know and love.

- **Dev/Test/Debug:**

When developers or administrators are working on VMs in the cloud, they often need a set of tools or utilities. Copying such utilities and tools to each VM can be a time consuming exercise. By mounting an Azure file share locally on the VMs, a developer and administrator can quickly access their tools and utilities, no copying required.

Key benefits

- **Shared access.** Azure file shares support the industry standard SMB protocol, meaning you can seamlessly replace your on-premises file shares with Azure file shares without worrying about application compatibility. Being able to share a file system across multiple machines, applications/instances is a significant advantage with Azure Files for applications that need shareability.
- **Fully managed.** Azure file shares can be created without the need to manage hardware or an OS. This means you don't have to deal with patching the server OS with critical security upgrades or replacing faulty hard disks.
- **Scripting and tooling.** PowerShell cmdlets and Azure CLI can be used to create, mount, and manage Azure file shares as part of the administration of Azure applications. You can create and manage Azure file shares using Azure portal and Azure Storage Explorer.
- **Resiliency.** Azure Files has been built from the ground up to be always available. Replacing on-premises file shares with Azure Files means you no longer have to wake up to deal with local power outages or network issues.
- **Familiar programmability.** Applications running in Azure can access data in the share via file [System I/O APIs](#). Developers can therefore leverage their existing code and skills to migrate existing applications. In addition to System IO APIs, you can use [Azure Storage Client Libraries](#) or the [Azure Storage REST API](#).

Next Steps

- [Create Azure file Share](#)
- [Connect and mount on Windows](#)
- [Connect and mount on Linux](#)
- [Connect and mount on macOS](#)

Quickstart: Create and manage Azure Files share with Windows virtual machines

12/10/2019 • 6 minutes to read • [Edit Online](#)

The article demonstrates the basic steps for creating and using an Azure Files share. In this quickstart, the emphasis is on quickly setting up an Azure Files share so you can experience how the service works. If you need more detailed instructions for creating and using Azure file shares in your own environment, see [Use an Azure file share with Windows](#).

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

Sign in to Azure

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

Prepare your environment

In this quickstart, you set up the following items:

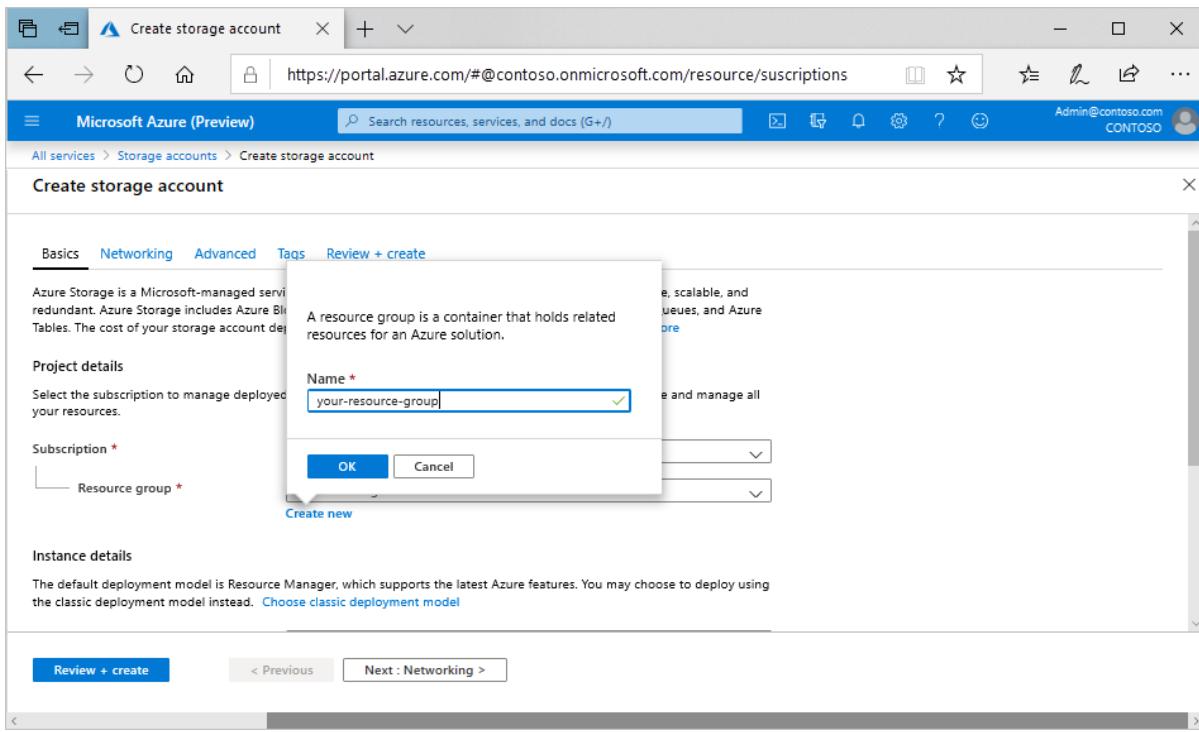
- An Azure storage account and an Azure file share
- A Windows Server 2016 Datacenter VM

Create a storage account

Before you can work with an Azure file share, you have to create an Azure storage account. A general-purpose v2 storage account provides access to all of the Azure Storage services: blobs, files, queues, and tables. The quickstart creates a general-purpose v2 storage account but, the steps to create any type of storage account are similar. A storage account can contain an unlimited number of shares. A share can store an unlimited number of files, up to the capacity limits of the storage account.

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 an Azure file share

Next, you create a file share.

1. When the Azure storage account deployment is complete, select **Go to resource**.
2. Select **Files** from the storage account pane.

The screenshot shows the Azure Storage Services page. It features four main sections: 'Blobs' (REST-based object storage for unstructured data), 'Files' (File shares that use the standard SMB 3.0 protocol, highlighted with a red border), 'Tables' (Tabular data storage), and 'Queues' (Effectively scale apps according to traffic). Each section includes a 'Learn more' link.

3. Select **File Share**.

The screenshot shows the 'qsstorageacct - Files' dashboard. It displays the storage account name and a 'File share' button, which is highlighted with a red box.

4. Name the new file share *qsfileshare* > enter "1" for the **Quota** > select **Create**. The quota can be a maximum of 5 TiB, but you only need 1 GiB for this quickstart.
5. Create a new txt file called *qsTestFile* on your local machine.
6. Select the new file share, then on the file share location, select **Upload**.

The screenshot shows the file share upload interface. The 'Upload' button is highlighted with a red box.

7. Browse to the location where you created your .txt file > select *qsTestFile.txt* > select **Upload**.

So far, you've created an Azure storage account and a file share with one file in it in Azure. Next you'll create the Azure VM with Windows Server 2016 Datacenter to represent the on-premises server in this quickstart.

Deploy a VM

1. Next, expand the menu on the left side of the portal and choose **Create a resource** in the upper left-hand corner of the Azure portal.
2. In the search box above the list of **Azure Marketplace** resources, search for and select **Windows Server 2016 Datacenter**, then choose **Create**.
3. In the **Basics** tab, under **Project details**, select the resource group you created for this quickstart.

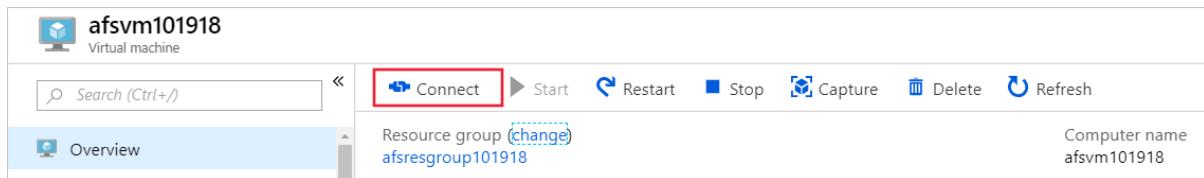
The screenshot shows the 'Create a virtual machine' wizard. The 'Basics' tab is selected. It includes fields for 'Subscription' (Visual Studio Enterprise), 'Resource group' (qsResourceGroup, highlighted with a red box), and 'Project details' (Subscription and Resource group selection).

4. Under **Instance details**, name the VM *qsVM*.
5. Leave the default settings for **Region**, **Availability options**, **Image**, and **Size**.
6. Under **Administrator account**, add *VMadmin* as the **Username** and enter a **Password** for the VM.
7. Under **Inbound port rules**, choose **Allow selected ports** and then select **RDP (3389)** and **HTTP** from the drop-down.
8. Select **Review + create**.
9. Select **Create**. Creating a new VM will take a few minutes to complete.
10. Once your VM deployment is complete, select **Go to resource**.

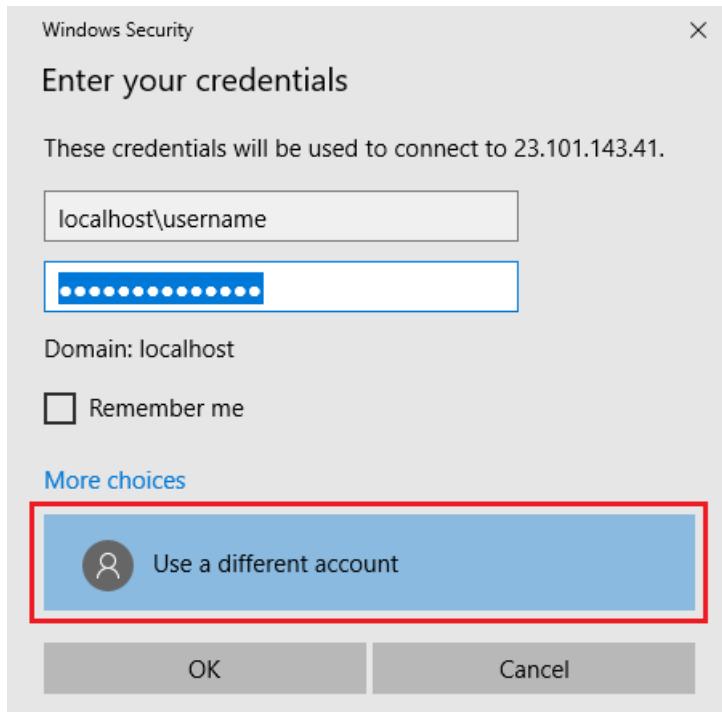
At this point, you've created a new virtual machine and attached a data disk. Now you need to connect to the VM.

Connect to your VM

1. Select **Connect** on the virtual machine properties page.



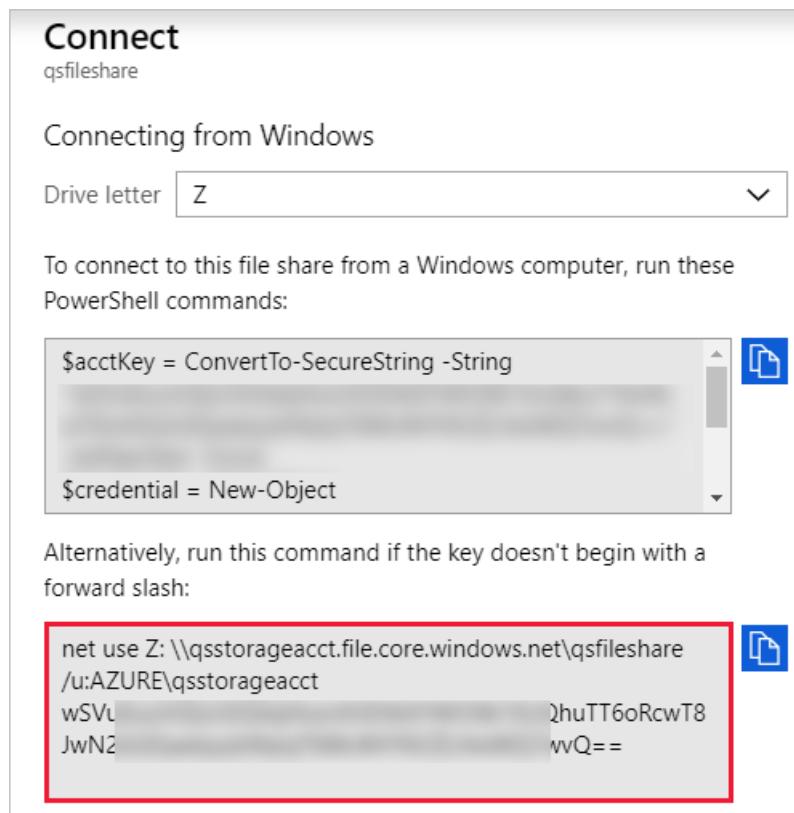
2. In the **Connect to virtual machine** page, keep the default options to connect by **IP address** over **port number** 3389 and select **Download RDP file**.
3. Open the downloaded RDP file and select **Connect** when prompted.
4. In the **Windows Security** window, select **More choices** and then **Use a different account**. Type the username as *localhost\username*, where <username> is the VM admin username you created for the virtual machine. Enter the password you created for the virtual machine, and then select **OK**.



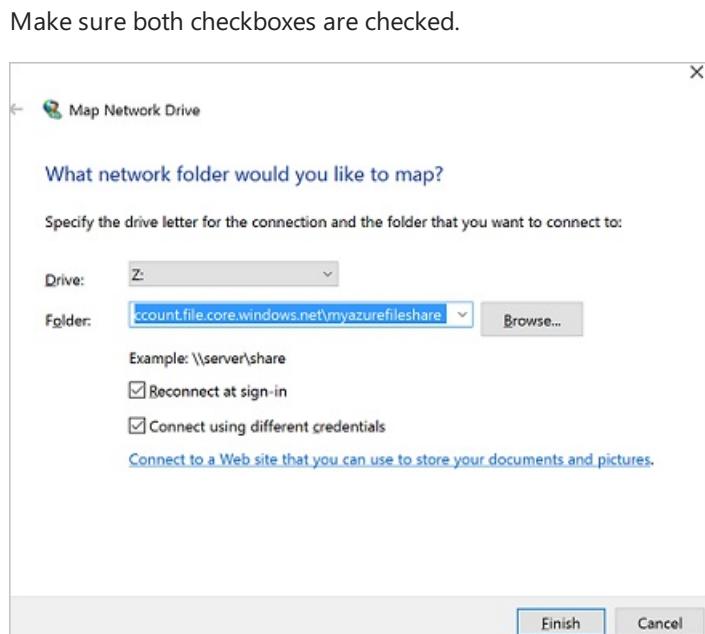
5. You may receive a certificate warning during the sign-in process. select **Yes** or **Continue** to create the connection.

Map the Azure file share to a Windows drive

1. In the Azure portal, navigate to the `qsfileshare` fileshare and select **Connect**.
2. Copy the contents of the second box and paste it in **Notepad**.



3. In the VM, open **File Explorer** and select **This PC** in the window. This selection will change the menus available on the ribbon. On the **Computer** menu, select **Map network drive**.
4. Select the drive letter and enter the UNC path. If you've followed the naming suggestions in this quickstart, copy `\qsstorageacct.file.core.windows.net\qsfileshare` from **Notepad**.



5. Select **Finish**.
6. In the **Windows Security** dialog box:
 - From Notepad, copy the storage account name prepended with AZURE\ and paste it in the

Windows Security dialog box as the username. If you've followed the naming suggestions in this quickstart, copy AZURE\qsstorageacct.

- From Notepad, copy the storage account key and paste it in the **Windows Security** dialog box as the password.

```
net use Z:\qsstorageacct.file.core.windows.net\qsfshare  
/u:AZURE\qsstorageacct  
wSVu QhuTT6oRcwT8  
JwN2 /vvQ==
```

Create a share snapshot

Now that you've mapped the drive, you can create a snapshot.

- In the portal, navigate to your file share and select **Create snapshot**.

The screenshot shows the Azure Storage File Share blade. At the top, there are several buttons: Connect, Upload, Add directory, Refresh, Delete share, Quota, View snapshots, and Create Snapshot. The 'Create Snapshot' button is highlighted with a red box. Below these buttons, there is a message: "Backup (Preview) is not enabled for this file share. Click here to enable backup." Underneath, there is a section labeled "Location: qsfshare". A search bar says "Search files by prefix". A table lists a single file: "NAME" (qsTestFile.txt), "TYPE" (File), and "SIZE" (0 B).

- In the VM, open the *qsTestfile.txt* and type "this file has been modified" > Save and close the file.
- Create another snapshot.

Browse a share snapshot

- On your file share, select **View snapshots**.
- On the **File share snapshots** pane, select the first snapshot in the list.

The screenshot shows the "File share snapshots" pane. At the top, there is a breadcrumb navigation: Dashboard > mystorageacct68 - Files > myfileshare > File share snapshots. Below this, there is a header "File share snapshots" and a sub-header "myfileshare". There are buttons for "Add snapshot", "Refresh", and "Delete". A table lists two snapshots:

NAME	DATE CREATED
2018-11-28T22:20:36.000000Z	11/28/2018, 2:20:36 PM
2018-11-28T22:17:59.000000Z	11/28/2018, 2:17:59 PM

- On the pane for that snapshot, select *qsTestFile.txt*.

Restore from a snapshot

- From the file share snapshot blade, right-click the *qsTestFile*, and select the **Restore** button.
- Select **Overwrite original file**.



3. In the VM, open the file. The unmodified version has been restored.

Delete a share snapshot

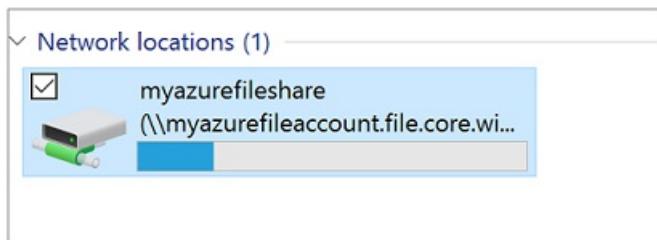
1. On your file share, select **View snapshots**.
2. On the **File share snapshots** pane, select the last snapshot in the list and click **Delete**.

NAME	DATE CREATED
2019-01-09T18:25:17.0000000Z	1/9/2019, 10:25:17 AM
2019-01-09T18:23:48.0000000Z	1/9/2019, 10:23:48 AM
2019-01-03T23:39:43.0000000Z	1/3/2019, 3:39:43 PM
<input checked="" type="checkbox"/> 2019-01-03T23:37:08.0000000Z	1/3/2019, 3:37:08 PM

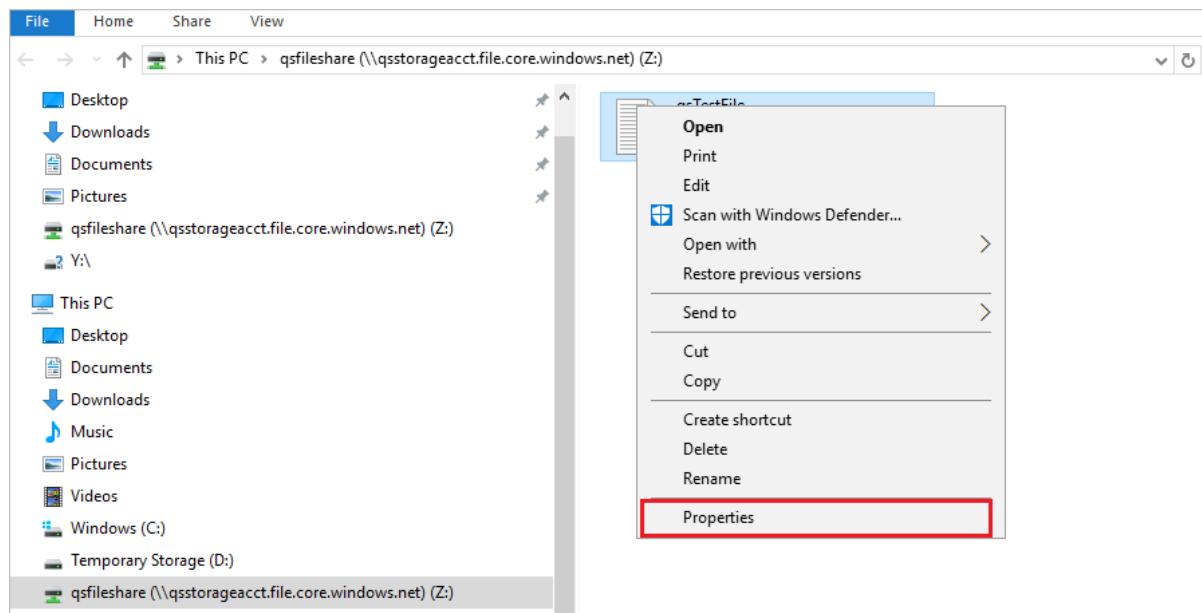
Use a share snapshot in Windows

Just like with on-premises VSS snapshots, you can view the snapshots from your mounted Azure file share by using the Previous Versions tab.

1. In File Explorer, locate the mounted share.

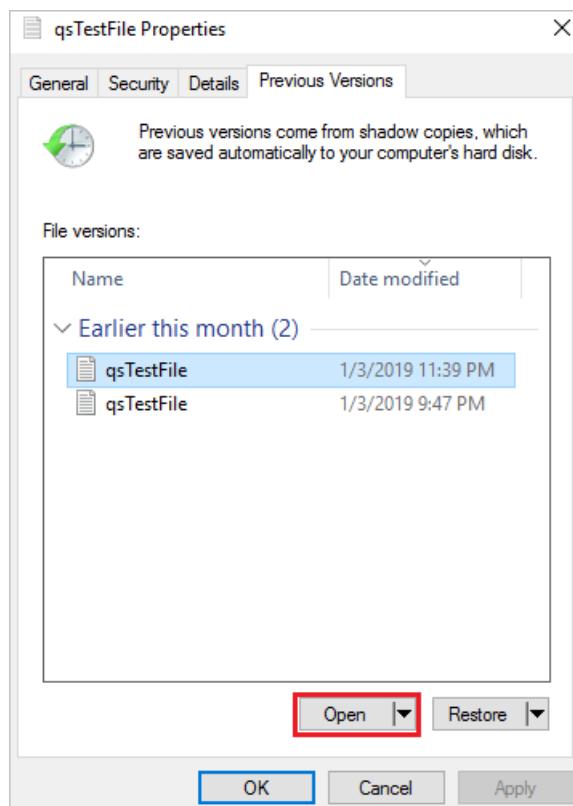


2. Select *qsTestFile.txt* and > right-click and select **Properties** from the menu.



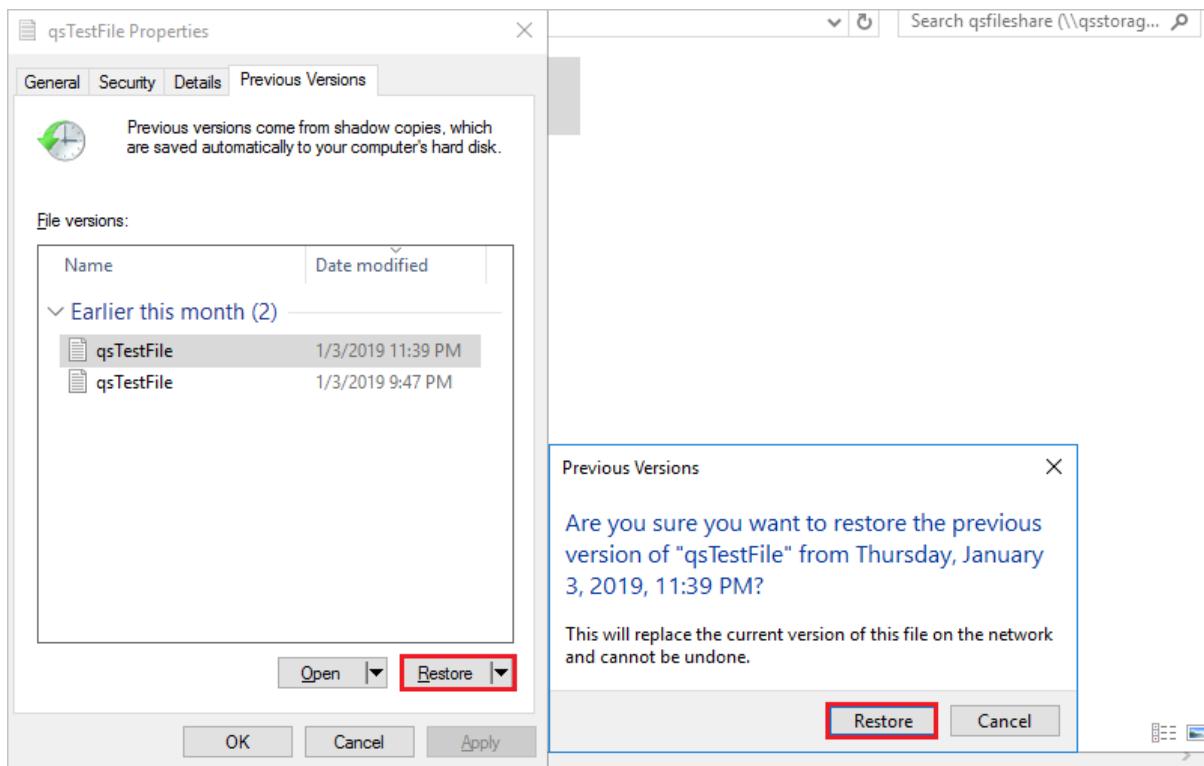
3. Select **Previous Versions** to see the list of share snapshots for this directory.

4. Select **Open** to open the snapshot.



Restore from a previous version

1. Select **Restore**. This action copies the contents of the entire directory recursively to the original location at the time the share snapshot was created.



Note: If your file has not changed, you will not see a previous version for that file because that file is the same version as the snapshot. This is consistent with how this works on a Windows file server.

Clean up resources

When you're done, you can delete the resource group. Deleting the resource group deletes the storage account, the Azure file share, and any other resources that you deployed inside the resource group.

1. In the left menu, select **Resource groups**.
2. Right-click the resource group, and then select **Delete resource group**. A window opens and displays a warning about the resources that will be deleted with the resource group.
3. Enter the name of the resource group, and then select **Delete**.

Next steps

[Use an Azure file share with Windows](#)

Quickstart: Create and manage Azure file shares with the Azure portal

9/24/2019 • 5 minutes to read • [Edit Online](#)

Azure Files is Microsoft's easy-to-use cloud file system. Azure file shares can be mounted in Windows, Linux, and macOS. This guide walks you through the basics of working with Azure file shares using the [Azure portal](#).

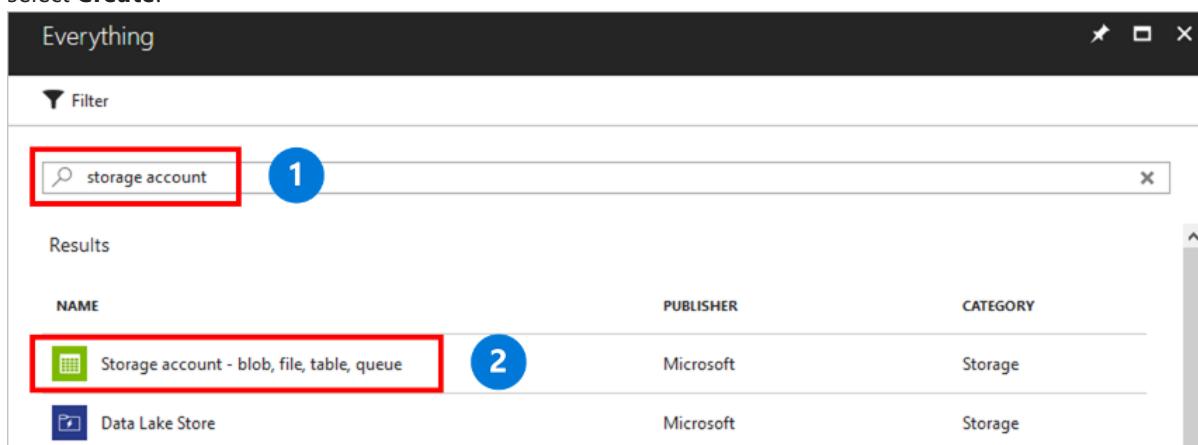
If you don't have an Azure subscription, create a [free account](#) before you begin.

Create a storage account

A storage account is a shared pool of storage in which you can deploy an Azure file share or other storage resources, such as blobs or queues. A storage account can contain an unlimited number of shares. A share can store an unlimited number of files, up to the capacity limits of the storage account.

To create a storage account:

1. In the left menu, select + to create a resource.
2. In the search box, enter **storage account**, select **Storage account - blob, file, table, queue**, and then select **Create**.



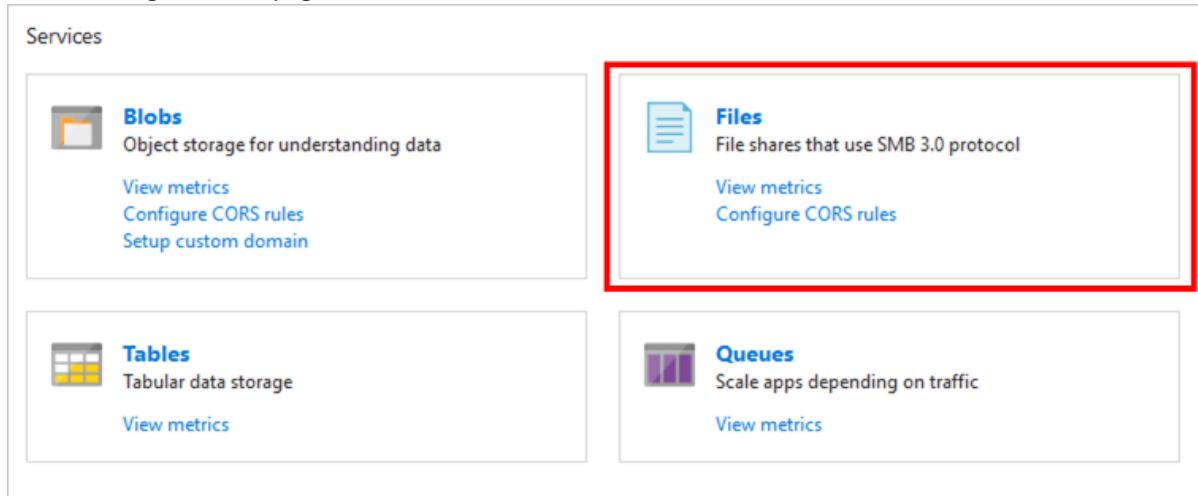
3. In **Name**, enter *mystorageacct* followed by a few random numbers, until you see a green check mark that indicates that it's a unique name. A storage account name must be all lowercase and globally unique. Make a note of your storage account name. You will use it later.
4. In **Deployment model**, leave the default value of **Resource Manager**. To learn more about the differences between Azure Resource Manager and the classic deployment model, see [Understand deployment models and the state of your resources](#).
5. In **Account kind**, select **StorageV2**. To learn more about the different kinds of storage accounts, see [Understand Azure storage accounts](#).
6. In **Performance**, keep the default value of **Standard storage**. Azure Files currently supports only standard storage; even if you select Azure Premium Storage, your file share is stored in standard storage.
7. In **Replication**, select **Locally redundant storage (LRS)**.
8. In **Secure transfer required**, we recommend that you always select **Enabled**. To learn more about this option, see [Understand encryption in-transit](#).

9. In **Subscription**, select the subscription that was used to create the storage account. If you have only one subscription, it should be the default.
10. In **Resource group**, select **Create new**. For the name, enter *myResourceGroup*.
11. In **Location**, select **East US**.
12. In **Virtual networks**, leave the default option as **Disabled**.
13. To make the storage account easier to find, select **Pin to dashboard**.
14. When you're finished, select **Create** to start the deployment.

Create an Azure file share

To create an Azure file share:

1. Select the storage account from your dashboard.
2. On the storage account page, in the **Services** section, select **Files**.



3. On the menu at the top of the **File service** page, click **File share**. The **New file share** page drops down.
4. In **Name** type *myshare*.
5. Click **OK** to create the Azure file share.

Share names need to be all lower case letters, numbers, and single hyphens but cannot start with a hyphen. For complete details about naming file shares and files, see [Naming and Referencing Shares, Directories, Files, and Metadata](#).

Use your Azure file share

Azure Files provides two methods of working with files and folders within your Azure file share: the industry standard [Server Message Block \(SMB\) protocol](#) and the [File REST protocol](#).

To mount a file share with SMB, see the following document based on your OS:

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

Using an Azure file share from the Azure portal

All requests made via the Azure portal are made with the File REST API enabling you to create, modify, and delete files and directories on clients without SMB access. It is possible to work directly with the File REST protocol (that

is, handcrafting REST HTTP calls yourself), but the most common way (beyond using the Azure portal) to use the File REST protocol is to use the [Azure PowerShell module](#), the [Azure CLI](#), or an Azure Storage SDK, all of which provide a nice wrapper around the File REST protocol in the scripting/programming language of your choice.

We expect most users of Azure Files will want to work with their Azure file share over the SMB protocol, as this allows them to use the existing applications and tools they expect to be able to use, but there are several reasons why it is advantageous to use the File REST API rather than SMB, such as:

- You need to make a quick change to your Azure file share from on-the-go, such as from a laptop without SMB access, tablet, or mobile device.
- You need to execute a script or application from a client which cannot mount an SMB share, such as on-premises clients, which do not have port 445 unblocked.
- You are taking advantage of serverless resources, such as [Azure Functions](#).

The following examples show how to use the Azure portal to manipulate your Azure file share with the File REST protocol.

Now that you have created an Azure file share, you can mount the file share with SMB on [Windows](#), [Linux](#), or [macOS](#). Alternatively, you can work with your Azure file share with the Azure portal.

Create a directory

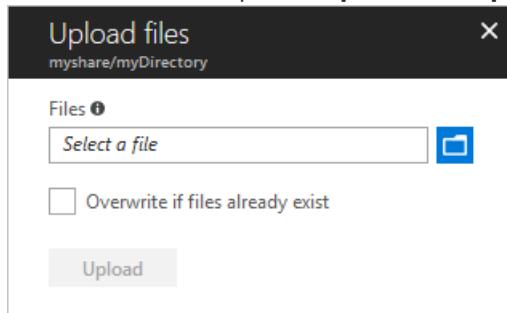
To create a new directory named *myDirectory* at the root of your Azure file share:

1. On the **File Service** page, select the **myshare** file share. The page for your file share opens.
2. On the menu at the top of the page, select **+ Add directory**. The **New directory** page drops down.
3. Type *myDirectory* and then click **OK**.

Upload a file

To demonstrate uploading a file, you first need to create or select a file to be uploaded. You may do this by whatever means you see fit. Once you've selected the file you would like to upload:

1. Click on the **myDirectory** directory. The **myDirectory** panel opens.
2. In the menu at the top, click **Upload**. The **Upload files** panel opens.



3. Click on the folder icon to open a window to browse your local files.
4. Select a file and then click **Open**.
5. In the **Upload files** page, verify the file name and then click **Upload**.
6. When finished, the file should appear in the list on the **myDirectory** page.

Download a file

You can download a copy of the file you uploaded by right-clicking on the file. After clicking the download button, the exact experience will depend on the operating system and browser you're using.

Clean up resources

When you're done, you can delete the resource group. Deleting the resource group deletes the storage account, the

Azure file share, and any other resources that you deployed inside the resource group.

1. In the left menu, select **Resource groups**.
2. Right-click the resource group, and then select **Delete resource group**. A window opens and displays a warning about the resources that will be deleted with the resource group.
3. Enter the name of the resource group, and then select **Delete**.

Next steps

[What is Azure Files?](#)

Quickstart: Create and manage an Azure file share with Azure PowerShell

2/25/2020 • 9 minutes to read • [Edit Online](#)

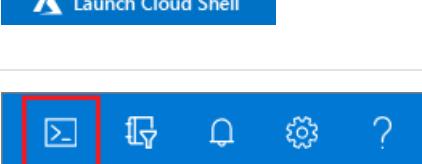
This guide walks you through the basics of working with [Azure file shares](#) with PowerShell. Azure file shares are just like other file shares, but stored in the cloud and backed by the Azure platform. Azure File shares support the industry standard SMB protocol and enable file sharing across multiple machines, applications, and instances.

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 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.

If you would like to install and use the PowerShell locally, this guide requires the Azure PowerShell module Az version 0.7 or later. To find out which version of the Azure PowerShell module you are running, execute `Get-Module -ListAvailable Az`. If you need to upgrade, see [Install Azure PowerShell module](#). If you are running PowerShell locally, you also need to run `Login-AzAccount` to login to your Azure account.

Create a resource group

A resource group is a logical container into which Azure resources are deployed and managed. If you don't already have an Azure resource group, you can create a new one with the [New-AzResourceGroup](#) cmdlet.

The following example creates a resource group named *myResourceGroup* in the West US 2 region:

```
$resourceGroupName = "myResourceGroup"
$region = "westus2"

New-AzResourceGroup ` 
    -Name $resourceGroupName ` 
    -Location $region | Out-Null
```

Create a storage account

A storage account is a shared pool of storage you can use to deploy Azure file shares. A storage account can contain an unlimited number of shares, and a share can store an unlimited number of files, up to the capacity limits of the storage account. This example creates a general purpose version 2 (GPv2 storage account), which can store standard Azure file shares or other storage resources such as blobs or queues, on hard-disk drive (HDD) rotational media. Azure Files also supports premium solid-state disk drives (SSDs); premium Azure file shares can be created in FileStorage storage accounts.

This example creates a storage account using the [New-AzStorageAccount](#) cmdlet. The storage account is named *mystorageaccount<random number>* and a reference to that storage account is stored in the variable **\$storageAcct**. Storage account names must be unique, so use [Get-Random](#) to append a number to the name to make it unique.

```
$storageAccountName = "mystorageacct$(Get-Random)"

$storageAcct = New-AzStorageAccount ` 
    -ResourceGroupName $resourceGroupName ` 
    -Name $storageAccountName ` 
    -Location $region ` 
    -Kind StorageV2 ` 
    -SkuName Standard_ZRS ` 
    -EnableLargeFileShare
```

NOTE

Shares greater than 5 TiB (up to a maximum of 100 TiB per share) are only available in locally redundant (LRS) and zone redundant (ZRS) storage accounts. To create a geo-redundant (GRS) or geo-zone-redundant (GZRS) storage account, remove the [-EnableLargeFileShare](#) parameter.

Create an Azure file share

Now you can create your first Azure file share. You can create a file share using the [New-AzRmStorageShare](#) cmdlet. This example creates a share named [myshare](#).

```
$shareName = "myshare"

New-AzRmStorageShare ` 
    -StorageAccount $storageAcct ` 
    -Name $shareName ` 
    -QuotaGiB 1024 | Out-Null
```

Share names need to be all lower-case letters, numbers, and single hyphens but cannot start with a hyphen. For complete details about naming file shares and files, see [Naming and Referencing Shares, Directories, Files, and Metadata](#).

Use your Azure file share

Azure Files provides two methods of working with files and folders within your Azure file share: the industry standard [Server Message Block \(SMB\) protocol](#) and the [File REST protocol](#).

To mount a file share with SMB, see the following document based on your OS:

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

Using an Azure file share with the File REST protocol

It is possible work with the File REST protocol directly (i.e. handcrafting REST HTTP calls yourself), but the most common way to use the File REST protocol is to use the Azure PowerShell module, the [Azure CLI](#), or an Azure Storage SDK, all of which provide a nice wrapper around the File REST protocol in the scripting/programming language of your choice.

In most cases, you will use your Azure file share over the SMB protocol, as this allows you to use the existing applications and tools you expect to be able to use, but there are several reasons why it is advantageous to use the File REST API rather than SMB, such as:

- You are browsing your file share from the PowerShell Cloud Shell (which cannot mount file shares over SMB).
- You are taking advantage of serverless resources, such as [Azure Functions](#).
- You are creating a value-add service that will interact with many Azure file shares, such as performing backup or antivirus scans.

The following examples show how to use the Azure PowerShell module to manipulate your Azure file share with the File REST protocol. The `-Context` parameter is used to retrieve the storage account key to perform the indicated actions against the file share. To retrieve the storage account key, you must have the RBAC role of `Owner` on the storage account.

Create directory

To create a new directory named `myDirectory` at the root of your Azure file share, use the [New-AzStorageDirectory](#) cmdlet.

```
New-AzStorageDirectory ` 
    -Context $storageAcct.Context ` 
    -ShareName $shareName ` 
    -Path "myDirectory"
```

Upload a file

To demonstrate how to upload a file using the [Set-AzStorageFileContent](#) cmdlet, we first need to create a file inside your PowerShell Cloud Shell's scratch drive to upload.

This example puts the current date and time into a new file on your scratch drive, then uploads the file to the file share.

```
# this expression will put the current date and time into a new file on your scratch drive
cd("~/CloudDrive/")
Get-Date | Out-File -FilePath "SampleUpload.txt" -Force

# this expression will upload that newly created file to your Azure file share
Set-AzStorageFileContent ` 
    -Context $storageAcct.Context ` 
    -ShareName $shareName ` 
    -Source "SampleUpload.txt" ` 
    -Path "myDirectory\SampleUpload.txt"
```

If you're running PowerShell locally, you should substitute `~/CloudDrive/` with a path that exists on your machine.

After uploading the file, you can use [Get-AzStorageFile](#) cmdlet to check to make sure that the file was uploaded to your Azure file share.

```
Get-AzStorageFile ` 
    -Context $storageAcct.Context ` 
    -ShareName $shareName ` 
    -Path "myDirectory\"
```

Download a file

You can use the [Get-AzStorageFileContent](#) cmdlet to download a copy of the file you just uploaded to the scratch drive of your Cloud Shell.

```
# Delete an existing file by the same name as SampleDownload.txt, if it exists because you've run this example before.
Remove-Item ` 
    -Path "SampleDownload.txt" ` 
    -Force ` 
    -ErrorAction SilentlyContinue

Get-AzStorageFileContent ` 
    -Context $storageAcct.Context ` 
    -ShareName $shareName ` 
    -Path "myDirectory\SampleUpload.txt" ` 
    -Destination "SampleDownload.txt"
```

After downloading the file, you can use the [Get-ChildItem](#) to see that the file has been downloaded to your PowerShell Cloud Shell's scratch drive.

```
Get-ChildItem | Where-Object { $_.Name -eq "SampleDownload.txt" }
```

Copy files

One common task is to copy files from one file share to another file share. To demonstrate this functionality, you can create a new share and copy the file you just uploaded over to this new share using the [Start-AzStorageFileCopy](#) cmdlet.

```
$otherShareName = "myshare2"

New-AzRmStorageShare ` 
    -StorageAccount $storageAcct ` 
    -Name $otherShareName ` 
    -QuotaGiB 1024 | Out-Null

New-AzStorageDirectory ` 
    -Context $storageAcct.Context ` 
    -ShareName $otherShareName ` 
    -Path "myDirectory2"

Start-AzStorageFileCopy ` 
    -Context $storageAcct.Context ` 
    -SrcShareName $shareName ` 
    -SrcFilePath "myDirectory\SampleUpload.txt" ` 
    -DestShareName $otherShareName ` 
    -DestFilePath "myDirectory2\SampleCopy.txt" ` 
    -DestContext $storageAcct.Context
```

Now, if you list the files in the new share, you should see your copied file.

```
Get-AzStorageFile ` 
    -Context $storageAcct.Context ` 
    -ShareName $otherShareName ` 
    -Path "myDirectory2"
```

While the `Start-AzStorageFileCopy` cmdlet is convenient for ad hoc file moves between Azure file shares, for migrations and larger data movements, we recommend `robocopy` on Windows and `rsync` on macOS and Linux. `robocopy` and `rsync` use SMB to perform the data movements instead of the FileREST API.

Create and manage share snapshots

One additional useful task you can do with an Azure file share is to create share snapshots. A snapshot preserves a point in time for an Azure file share. Share snapshots are similar to operating system technologies you may already be familiar with such as:

- [Volume Shadow Copy Service \(VSS\)](#) for Windows file systems such as NTFS and ReFS.
- [Logical Volume Manager \(LVM\)](#) snapshots for Linux systems.
- [Apple File System \(APFS\)](#) snapshots for macOS.

You can create a share snapshot for a share by using the `Snapshot` method on PowerShell object for a file share, which is retrieved with the `Get-AzStorageShare` cmdlet.

```
$share = Get-AzStorageShare -Context $storageAcct.Context -Name $shareName
$snapshot = $share.Snapshot()
```

Browse share snapshots

You can browse the contents of the share snapshot by passing the snapshot reference (`$snapshot`) to the `-Share` parameter of the `Get-AzStorageFile` cmdlet.

```
Get-AzStorageFile -Share $snapshot
```

List share snapshots

You can see the list of snapshots you've taken for your share with the following command.

```
Get-AzStorageShare ` 
    -Context $storageAcct.Context | ` 
    Where-Object { $_.Name -eq $shareName -and $_.IsSnapshot -eq $true }
```

Restore from a share snapshot

You can restore a file by using the `Start-AzStorageFileCopy` command we used before. For the purposes of this quickstart, we'll first delete our `SampleUpload.txt` file we previously uploaded so we can restore it from the snapshot.

```
# Delete SampleUpload.txt
Remove-AzStorageFile `-
    -Context $storageAcct.Context `-
    -ShareName $shareName `-
    -Path "myDirectory\SampleUpload.txt"

# Restore SampleUpload.txt from the share snapshot
Start-AzStorageFileCopy `-
    -SrcShare $snapshot `-
    -SrcFilePath "myDirectory\SampleUpload.txt" `-
    -DestContext $storageAcct.Context `-
    -DestShareName $shareName `-
    -DestFilePath "myDirectory\SampleUpload.txt"
```

Delete a share snapshot

You can delete a share snapshot by using the [Remove-AzStorageShare](#) cmdlet, with the variable containing the `$snapshot` reference to the `-Share` parameter.

```
Remove-AzStorageShare `-
    -Share $snapshot `-
    -Confirm:$false `-
    -Force
```

Clean up resources

When you are done, you can use the [Remove-AzResourceGroup](#) cmdlet to remove the resource group and all related resources.

```
Remove-AzResourceGroup -Name myResourceGroup
```

You can alternatively remove resources one by one:

- To remove the Azure file shares we created for this quickstart.

```
Get-AzRmStorageShare -StorageAccount $storageAcct | Remove-AzRmStorageShare -Force
```

NOTE

You must delete all the share snapshots for the Azure file shares you created before deleting the Azure file share.

- To remove the storage account itself (this will implicitly remove the Azure file shares we created as well as any other storage resources you may have created such as an Azure Blob storage container).

```
Remove-AzStorageAccount `-
    -ResourceGroupName $storageAcct.ResourceGroupName `-
    -Name $storageAcct.StorageAccountName
```

Next steps

[What is Azure Files?](#)

Quickstart: Create and manage Azure file shares using Azure CLI

2/25/2020 • 9 minutes to read • [Edit Online](#)

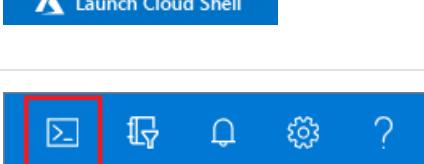
This guide walks you through the basics of working with [Azure file shares](#) with the Azure CLI. Azure file shares are just like other file shares, but stored in the cloud and backed by the Azure platform. Azure File shares support the industry standard SMB protocol and enable file sharing across multiple machines, applications, and instances.

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 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.

If you decide to install and use Azure CLI locally, for the steps in this article, you must be running Azure CLI version 2.0.4 or later. Run **az --version** to find your Azure CLI version. If you need to install or upgrade, see [Install Azure CLI 2.0](#).

By default, Azure CLI commands return JavaScript Object Notation (JSON). JSON is the standard way to send and receive messages from REST APIs. To facilitate working with JSON responses, some of the examples in this article use the *query* parameter on Azure CLI commands. This parameter uses the [JMESPath query language](#) to parse JSON. To learn more about how to use the results of Azure CLI commands by following the JMESPath query language, see the [JMESPath tutorial](#).

Create a resource group

A resource group is a logical container in which Azure resources are deployed and managed. If you don't already have an Azure resource group, you can use the [az group create](#) command to create one.

The following example creates a resource group named *myResourceGroup* in the *West US 2* location:

```
export resourceGroupName="myResourceGroup"
region="westus2"

az group create \
--name $resourceGroupName \
--location $region \
--output none
```

Create a storage account

A storage account is a shared pool of storage in which you can deploy Azure file shares or other storage resources, such as blobs or queues. A storage account can contain an unlimited number of file shares. A share can store an unlimited number of files, up to the capacity limits of the storage account.

The following example creates a storage account using the [az storage account create](#) command. Storage account names must be unique, so use `$RANDOM` to append a number to the name to make it unique.

```
export storageAccountName="mystorageacct$RANDOM"

az storage account create \
--resource-group $resourceGroupName \
--name $storageAccountName \
--location $region \
--kind StorageV2 \
--sku Standard_LRS \
--enable-large-file-share \
--output none
```

NOTE

Shares greater than 5 TiB (up to a maximum of 100 TiB per share) are only available in locally redundant (LRS) and zone redundant (ZRS) storage accounts. To create a geo-redundant (GRS) or geo-zone-redundant (GZRS) storage account, remove the `--enable-large-file-share` parameter.

Get the storage account key

Storage account keys control access to resources in a storage account. The keys are automatically created when you create a storage account. You can get the storage account keys for your storage account by using the [az storage account keys list](#) command:

```
export storageAccountKey=$(az storage account keys list \
--resource-group $resourceGroupName \
--account-name $storageAccountName \
--query "[0].value" | tr -d "'")
```

Create an Azure file share

Now, you can create your first Azure file share. Create file shares by using the [az storage share create](#) command. This example creates an Azure file share named *myshare*:

```
shareName="myshare"

az storage share create \
--account-name $storageAccountName \
--account-key $storageAccountKey \
--name $shareName \
--quota 1024 \
--output none
```

Share names can contain only lowercase letters, numbers, and single hyphens (but they can't start with a hyphen). For complete details about naming file shares and files, see [Naming and referencing shares, directories, files, and metadata](#).

Use your Azure file share

Azure Files provides two methods of working with files and folders within your Azure file share: the industry standard [Server Message Block \(SMB\) protocol](#) and the [File REST protocol](#).

To mount a file share with SMB, see the following document based on your OS:

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

Using an Azure file share with the File REST protocol

It is possible work directly with the File REST protocol directly (handcrafting REST HTTP calls yourself), but the most common way to use the File REST protocol is to use the Azure CLI, the [Azure PowerShell module](#), or an Azure Storage SDK, all of which provide a nice wrapper around the File REST protocol in the scripting/programming language of your choice.

We expect most uses of Azure Files will want to work with their Azure file share over the SMB protocol, as this allows them to use the existing applications and tools they expect to be able to use, but there are several reasons why it is advantageous to use the File REST API rather than SMB, such as:

- You are browsing your file share from the Azure Bash Cloud Shell (which cannot mount file shares over SMB).
- You are taking advantage of serverless resources, such as [Azure Functions](#).
- You are creating a value-add service that will interact with many Azure file shares, such as performing backup or antivirus scans.

The following examples show how to use the Azure CLI to manipulate your Azure file share with the File REST protocol.

Create a directory

To create a new directory named *myDirectory* at the root of your Azure file share, use the

```
az storage directory create
```

```
az storage directory create \
--account-name $storageAccountName \
--account-key $storageAccountKey \
--share-name $shareName \
--name "myDirectory" \
--output none
```

Upload a file

To demonstrate how to upload a file by using the `az storage file upload` command, first create a file to upload on the Cloud Shell scratch drive. In the following example, you create and then upload the file:

```
cd ~/clouddrive/  
date > SampleUpload.txt  
  
az storage file upload \  
--account-name $storageAccountName \  
--account-key $storageAccountKey \  
--share-name $shareName \  
--source "SampleUpload.txt" \  
--path "myDirectory/SampleUpload.txt"
```

If you're running Azure CLI locally, substitute `~/clouddrive` with a path that exists on your machine.

After you upload the file, you can use the `az storage file list` command to make sure that the file was uploaded to your Azure file share:

```
az storage file list \  
--account-name $storageAccountName \  
--account-key $storageAccountKey \  
--share-name $shareName \  
--path "myDirectory" \  
--output table
```

Download a file

You can use the `az storage file download` command to download a copy of the file that you uploaded to the Cloud Shell scratch drive:

```
# Delete an existing file by the same name as SampleDownload.txt, if it exists, because you've run this  
example before  
rm -f SampleDownload.txt  
  
az storage file download \  
--account-name $storageAccountName \  
--account-key $storageAccountKey \  
--share-name $shareName \  
--path "myDirectory/SampleUpload.txt" \  
--dest "SampleDownload.txt" \  
--output none
```

Copy files

A common task is to copy files from one file share to another file share. To demonstrate this functionality, create a new share. Copy the file that you uploaded to this new share by using the `az storage file copy` command:

```

otherShareName="myshare2"

az storage share create \
--account-name $storageAccountName \
--account-key $storageAccountKey \
--name $otherShareName \
--quota 1024 \
--output none

az storage directory create \
--account-name $storageAccountName \
--account-key $storageAccountKey \
--share-name $otherShareName \
--name "myDirectory2" \
--output none

az storage file copy start \
--account-name $storageAccountName \
--account-key $storageAccountKey \
--source-share $shareName \
--source-path "myDirectory/SampleUpload.txt" \
--destination-share $otherShareName \
--destination-path "myDirectory2/SampleCopy.txt"

```

Now, if you list the files in the new share, you should see your copied file:

```

az storage file list \
--account-name $storageAccountName \
--account-key $storageAccountKey \
--share-name $otherShareName \
--path "myDirectory2" \
--output table

```

While the `az storage file copy start` command is convenient for file moves between Azure file shares, for migrations and larger data movements, we recommend `rsync` on macOS and Linux and `robocopy` on Windows. `rsync` and `robocopy` use SMB to perform the data movements instead of the FileREST API.

Create and manage share snapshots

Another useful task that you can do with an Azure file share is create share snapshots. A snapshot preserves a point-in-time copy of an Azure file share. Share snapshots are similar to some operating system technologies that you might already be familiar with:

- [Logical Volume Manager \(LVM\)](#) snapshots for Linux systems.
- [Apple File System \(APFS\)](#) snapshots for macOS.
- [Volume Shadow Copy Service \(VSS\)](#) for Windows file systems, such as NTFS and ReFS.

You can create a share snapshot by using the `az storage share snapshot` command:

```

snapshot=$(az storage share snapshot \
--account-name $storageAccountName \
--account-key $storageAccountKey \
--name $shareName \
--query "snapshot" | tr -d '')

```

Browse share snapshot contents

You can browse the contents of a share snapshot by passing the time stamp of the share snapshot that you captured in the `$snapshot` variable to the `az storage file list` command:

```
az storage file list \
--account-name $storageAccountName \
--account-key $storageAccountKey \
--share-name $shareName \
--snapshot $snapshot \
--output table
```

List share snapshots

To see the list of snapshots that you've taken for your share, use the following command:

```
az storage share list \
--account-name $storageAccountName \
--account-key $storageAccountKey \
--include-snapshot \
--query "[? name== '$shareName' && snapshot!=null].snapshot" \
--output tsv
```

Restore from a share snapshot

You can restore a file by using the `az storage file copy start` command that you used earlier. First, delete the SampleUpload.txt file that you uploaded, so you can restore it from the snapshot:

```
# Delete SampleUpload.txt
az storage file delete \
--account-name $storageAccountName \
--account-key $storageAccountKey \
--share-name $shareName \
--path "myDirectory/SampleUpload.txt" \
--output none

# Build the source URI for a snapshot restore
URI=$(az storage account show \
--resource-group $resourceGroupName \
--name $storageAccountName \
--query "primaryEndpoints.file" | tr -d "'")

URI=$URI$shareName"/myDirectory/SampleUpload.txt?sharesnapshot="$snapshot

# Restore SampleUpload.txt from the share snapshot
az storage file copy start \
--account-name $storageAccountName \
--account-key $storageAccountKey \
--source-uri $URI \
--destination-share $shareName \
--destination-path "myDirectory/SampleUpload.txt"
```

Delete a share snapshot

You can delete a share snapshot by using the `az storage share delete` command. Use the variable that contains the `$SNAPSHOT` reference to the `--snapshot` parameter:

```
az storage share delete \
--account-name $storageAccountName \
--account-key $storageAccountKey \
--name $shareName \
--snapshot $snapshot \
--output none
```

Clean up resources

When you are done, you can use the `az group delete` command to remove the resource group and all related resources:

```
az group delete --name $resourceGroupName
```

Alternatively, you can remove resources individually.

- To remove the Azure file shares that you created for this article:

```
az storage share list \  
    --account-name $storageAccountName \  
    --account-key $storageAccountKey \  
    --query "[].name" \  
    --output tsv | \  
xargs -L1 bash -ec ' \  
    az storage share delete \  
    --account-name "$storageAccountName" \  
    --account-key "$storageAccountKey" \  
    --name $0 \  
    --delete-snapshots include \  
    --output none'
```

- To remove the storage account itself. (This implicitly removes the Azure file shares that you created, and any other storage resources that you might have created, such as an Azure Blob storage container.)

```
az storage account delete \  
    --resource-group $resourceGroupName \  
    --name $storageAccountName \  
    --yes
```

Next steps

[What is Azure Files?](#)

Quickstart: Create and manage Azure file shares with Azure Storage Explorer

12/10/2019 • 5 minutes to read • [Edit Online](#)

This guide walks you through the basics of working with [Azure file shares](#) with the Azure Storage Explorer. Azure file shares are just like other file shares, but stored in the cloud and backed by the Azure platform. Azure File shares support the industry standard SMB protocol and enable file sharing across multiple machines, applications, and instances.

The Azure Storage Explorer is a popular client tool that's available for Windows, macOS, and Linux. You can use Storage Explorer to manage Azure file shares and other storage resources.

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

Prerequisites

This quickstart requires Storage Explorer to be installed. To download and install it, go to [Azure Storage Explorer](#).

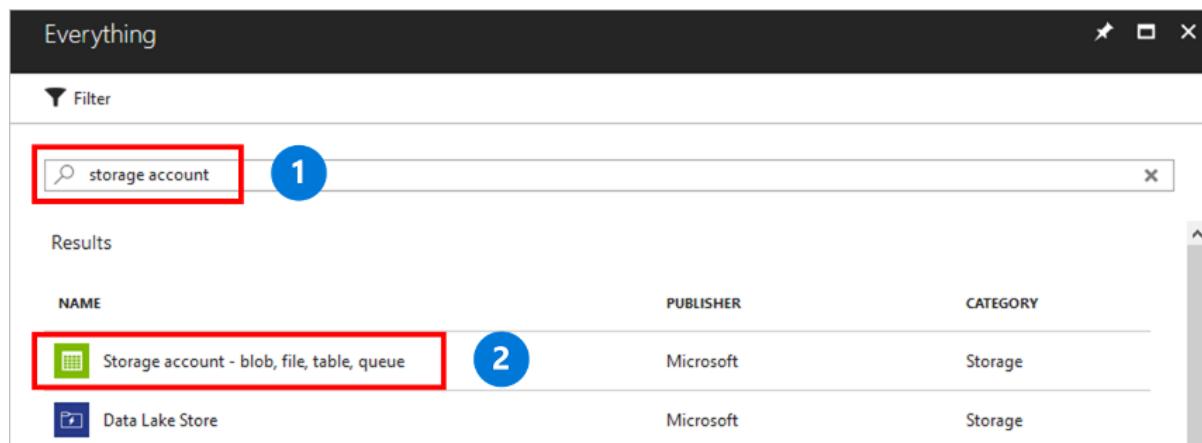
Create a storage account

You can't use Storage Explorer to create new resources. For the purposes of this demo, create the storage account in the [Azure portal](#).

A storage account is a shared pool of storage in which you can deploy an Azure file share or other storage resources, such as blobs or queues. A storage account can contain an unlimited number of shares. A share can store an unlimited number of files, up to the capacity limits of the storage account.

To create a storage account:

1. In the left menu, select + to create a resource.
2. In the search box, enter **storage account**, select **Storage account - blob, file, table, queue**, and then select **Create**.



The screenshot shows the Azure portal search interface. At the top, there is a search bar with the text "storage account". Below the search bar, a "Results" section displays two items in a table format:

NAME	PUBLISHER	CATEGORY
Storage account - blob, file, table, queue	Microsoft	Storage
Data Lake Store	Microsoft	Storage

3. In **Name**, enter *mystorageacct* followed by a few random numbers, until you see a green check mark that indicates that it's a unique name. A storage account name must be all lowercase and globally unique. Make a note of your storage account name. You will use it later.
4. In **Deployment model**, leave the default value of **Resource Manager**. To learn more about the differences between Azure Resource Manager and the classic deployment model, see [Understand deployment models](#)

and the state of your resources.

5. In **Account kind**, select **StorageV2**. To learn more about the different kinds of storage accounts, see [Understand Azure storage accounts](#).
6. In **Performance**, keep the default value of **Standard storage**. Azure Files currently supports only standard storage; even if you select Azure Premium Storage, your file share is stored in standard storage.
7. In **Replication**, select **Locally redundant storage (LRS)**.
8. In **Secure transfer required**, we recommend that you always select **Enabled**. To learn more about this option, see [Understand encryption in-transit](#).
9. In **Subscription**, select the subscription that was used to create the storage account. If you have only one subscription, it should be the default.
10. In **Resource group**, select **Create new**. For the name, enter *myResourceGroup*.
11. In **Location**, select **East US**.
12. In **Virtual networks**, leave the default option as **Disabled**.
13. To make the storage account easier to find, select **Pin to dashboard**.
14. When you're finished, select **Create** to start the deployment.

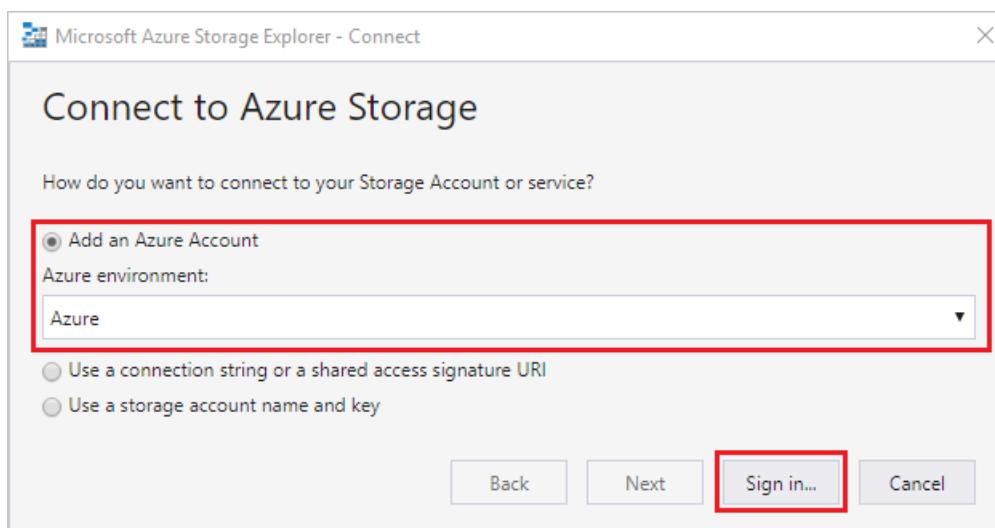
Connect Storage Explorer to Azure resources

When you first start Storage Explorer, the **Microsoft Azure Storage Explorer - Connect** window appears.

Storage Explorer provides several ways to connect to storage accounts:

- **Sign in by using your Azure account:** You can sign in by using the user credentials for your organization or your Microsoft account.
- **Connect to a specific storage account by using a connection string or SAS token:** A connection string is a special string that contains a storage account name and storage account key/SAS token. With the token, Storage Explorer directly accesses the storage account (rather than simply seeing all the storage accounts in an Azure account). To learn more about connection strings, see [Configure Azure storage connection strings](#).
- **Connect to a specific storage account by using a storage account name and key:** Use the storage account name and the key for your storage account to connect to Azure storage.

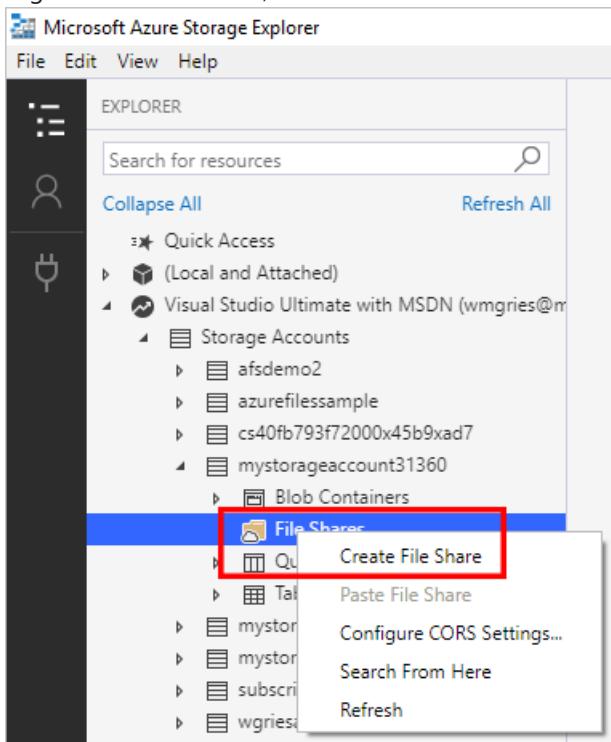
For the purposes of this quickstart, sign in by using your Azure account. Select **Add an Azure Account**, and then select **Sign in**. Follow the prompts to sign in to your Azure account.



Create a file share

To create your first Azure file share in the `storageacct<random number>` storage account:

1. Expand the storage account that you created.
2. Right-click **File Shares**, and then select **Create File Share**.



3. For the file share, enter *myshare*, and then press Enter.

Share names can contain only lowercase letters, numbers, and single hyphens (but they can't start with a hyphen). For complete details about naming file shares and files, see [Naming and referencing shares, directories, files, and metadata](#).

After the file share is created, a tab for your file share opens in the right pane.

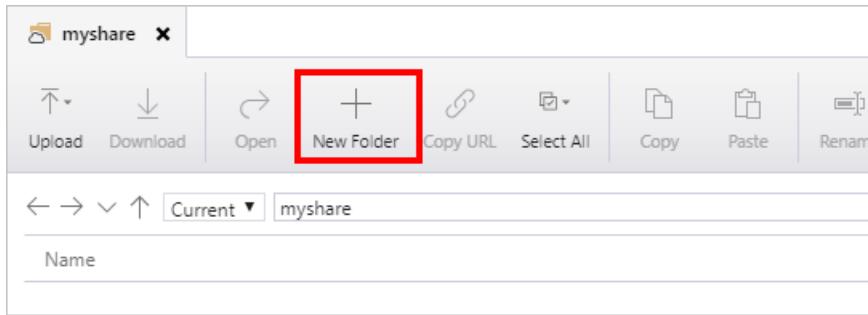
Use your Azure file share

Now that you have created an Azure file share, you can mount the file share with SMB on [Windows](#), [Linux](#), or [macOS](#). Alternatively, you can work with your Azure file share by using Azure Storage Explorer. The advantage of using Azure Storage Explorer instead of mounting the file share by using SMB is that all requests that are made with Azure Storage Explorer are made by using the File REST API. You can use the File REST API to create, modify, and delete files and directories on clients that don't have SMB access.

Create a directory

Adding a directory provides a hierarchical structure for managing your file share. You can create multiple levels in your directory. But, you must ensure that parent directories exist before you create subdirectories. For example, for the path *myDirectory/mySubDirectory*, you must create the directory *myDirectory* first. Then, you can create *mySubDirectory*.

1. On the tab for the file share, on the top menu, select the **New Folder** button. The **Create New Directory**



pane opens.

2. For the directory name, enter *myDirectory*, and then select **OK**.

The *myDirectory* directory is listed on the tab for the *myshare* file share.

Upload a file

You can upload a file from your local machine to the new directory in your file share. You can upload an entire folder or a single file.

1. In the top menu, select **Upload**. This gives you the option to upload a folder or a file.
2. Select **Upload File**, and then select a file to upload from your local machine.
3. In **Upload to a directory**, enter *myDirectory*, and then select **Upload**.

When you are finished, the file appears in the list in the *myDirectory* pane.

Download a file

To download a copy of a file from your file share, right-click the file, and then select **Download**. Choose where you want to put the file on your local machine, and then select **Save**.

The progress of the download appears in the **Activities** pane at the bottom of the window.

Clean up resources

You can't use Storage Explorer to remove resources. To clean up from this quickstart, you can use the [Azure portal](#).

When you're done, you can delete the resource group. Deleting the resource group deletes the storage account, the Azure file share, and any other resources that you deployed inside the resource group.

1. In the left menu, select **Resource groups**.
2. Right-click the resource group, and then select **Delete resource group**. A window opens and displays a warning about the resources that will be deleted with the resource group.
3. Enter the name of the resource group, and then select **Delete**.

Next steps

[What is Azure Files?](#)

Tutorial: Extend Windows file servers with Azure File Sync

2/25/2020 • 11 minutes to read • [Edit Online](#)

The article demonstrates the basic steps for extending the storage capacity of a Windows server by using Azure File Sync. Although the tutorial features Windows Server as an Azure virtual machine (VM), you would typically do this process for your on-premises servers. You can find instructions for deploying Azure File Sync in your own environment in the [Deploy Azure File Sync](#) article.

- Deploy the Storage Sync Service
- Prepare Windows Server to use with Azure File Sync
- Install the Azure File Sync agent
- Register Windows Server with the Storage Sync Service
- Create a sync group and a cloud endpoint
- Create a server endpoint

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

Sign in to Azure

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

Prepare your environment

For this tutorial, you need to do the following before you can deploy Azure File Sync:

- Create an Azure storage account and file share
- Set up a Windows Server 2016 Datacenter VM
- Prepare the Windows Server VM for Azure File Sync

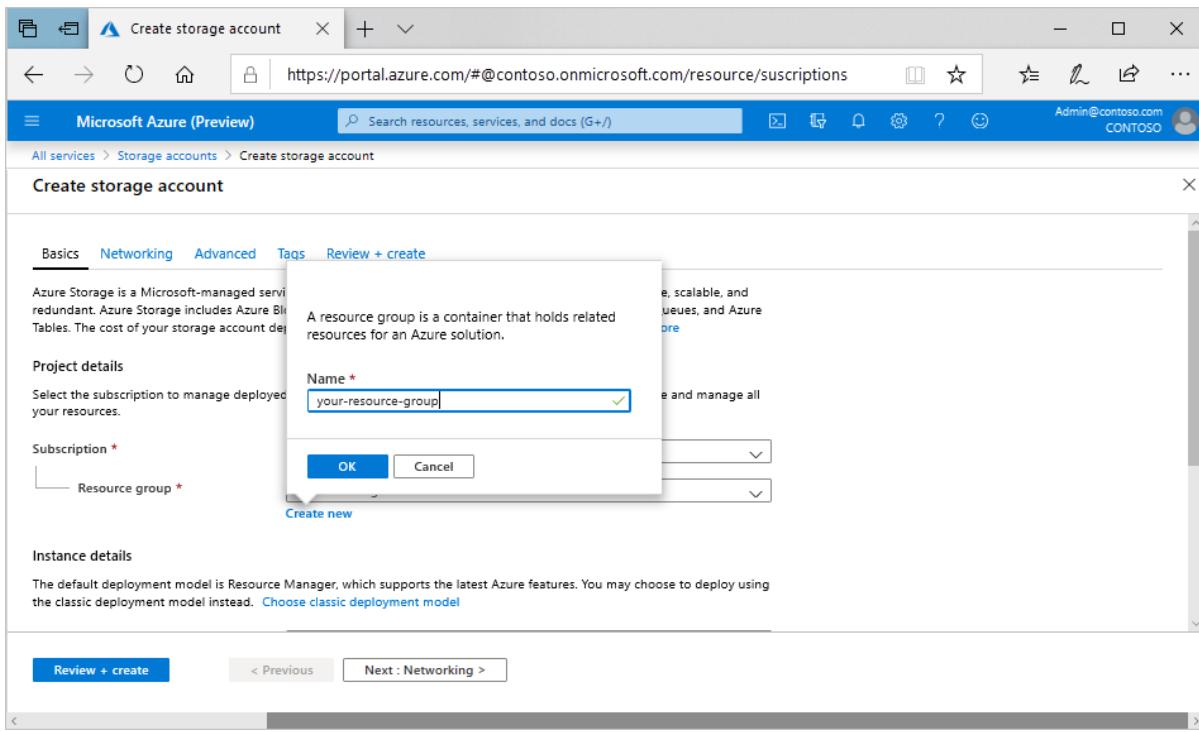
Create a folder and .txt file

On your local computer, create a new folder named *FilesToSync* and add a text file named *mytestdoc.txt*. You'll upload that file to the file share later in this tutorial.

Create a storage account

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 file share

After you deploy an Azure storage account, you create a file share.

1. In the Azure portal, select **Go to resource**.
2. Select **Files** from the storage account pane.

The screenshot shows the Azure Storage Account Overview page for 'afsstoracct101918'. On the left, there's a navigation menu with options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Events, Storage Explorer (preview), Settings, and Access keys. The main area is titled 'Essentials' and contains four service cards: 'Blobs' (REST-based object storage for unstructured data), 'Files' (File shares that use the standard SMB 3.0 protocol, which is highlighted with a red box), 'Tables' (Tabular data storage), and 'Queues' (Effectively scale apps according to traffic). At the top right, there are buttons for 'Open in Explorer', 'Move', 'Delete', and 'Refresh'.

3. Select + File Share.

The screenshot shows the 'Files' section of the 'afsstoracct101918' storage account. It includes a search bar, a 'File share' creation button (highlighted with a red box), a 'Refresh' button, and a 'Storage account' dropdown set to 'afsstoracct101918'. Below these are sections for 'Search file shares by prefix' (with 'NAME' and 'afsfileshare' entered) and a table with one row. The left sidebar has links for Overview, Activity log, Access control (IAM), and Tags.

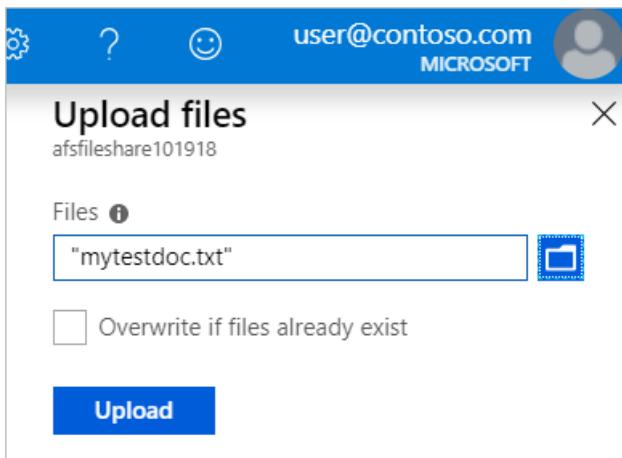
4. Name the new file share *afsfileshare*. Enter "1" for the **Quota**, and then select **Create**. The quota can be a maximum of 5 TiB, but you only need 1 GB for this tutorial.

The screenshot shows the 'File share' creation dialog. It has fields for 'Name' (containing 'afsfileshare101918') and 'Quota' (containing '1' followed by a dropdown menu showing 'GB'). At the bottom are 'Create' and 'Discard' buttons, with 'Create' being highlighted.

5. Select the new file share. On the file share location, select **Upload**.



6. Browse to the *FilesToSync* folder where you created your .txt file, select *mytestdoc.txt* and select **Upload**.



At this point, you've created a storage account and a file share with one file in it. Next, you deploy an Azure VM with Windows Server 2016 Datacenter to represent the on-premises server in this tutorial.

Deploy a VM and attach a data disk

1. Go to the Azure portal and expand the menu on the left. Choose **Create a resource** in the upper left-hand corner.
2. In the search box above the list of **Azure Marketplace** resources, search for **Windows Server 2016 Datacenter** and select it in the results. Choose **Create**.
3. Go to the **Basics** tab. Under **Project details**, select the resource group you created for this tutorial.

Create a virtual machine

Basics **Disks** **Networking** **Management** **Guest config** **Tags** **Review + create**

Create a virtual machine that runs Linux or Windows. Select an image from Azure marketplace or use your own customized image. Complete the Basics tab then Review + create to provision a virtual machine with default parameters or review each tab for full customization.

Looking for classic VMs? [Create VM from Azure Marketplace](#)

PROJECT DETAILS

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

* Subscription [Visual Studio Enterprise](#)

* Resource group [afsresgroup101918](#) [Create new](#)

4. Under **Instance details**, provide a VM name. For example, use *myVM*.
5. Don't change the default settings for **Region**, **Availability options**, **Image**, and **Size**.
6. Under **Administrator account**, provide a **Username** and **Password** for the VM.
7. Under **Inbound port rules**, choose **Allow selected ports** and then select **RDP (3389)** and **HTTP** from the drop-down menu.
8. Before you create the VM, you need to create a data disk.
 - a. Select **Next:Disks**.
 - b. On the **Disks** tab, under **Disk options**, leave the defaults.

- c. Under **DATA DISKS**, select **Create and attach a new disk**.
- d. Use the default settings except for **Size (GiB)**, which you can change to **1 GB** for this tutorial.

Create a new disk

Create a new disk to store applications and data on your VM. Disk pricing varies based on factors including disk size, storage type, and number of transactions. [Learn more about Azure Managed Disks](#)

* Disk type <small>i</small>	Premium SSD
* Name	afsvm101918_DataDisk_0
* Size (GiB) <small>i</small>	1
* Source type <small>i</small>	None (empty disk)

- e. Select **OK**.
 - 9. Select **Review + create**.
 - 10. Select **Create**.
- You can select the **Notifications** icon to watch the **Deployment progress**. Creating a new VM might take a few minutes to complete.

11. After your VM deployment is complete, select **Go to resource**.

CreateVm-MicrosoftWindowsServer.WindowsServer-201-20181019135159 - Overview

Deployment

Search (Ctrl+ /) Delete Cancel Redeploy Refresh

Overview Your deployment is complete Go to resource Deployment name: CreateVm-MicrosoftWindowsServer.WindowsServer-201-20181019135159 Subscription: Visual Studio Enterprise Resource group: afsresgroup101918

Outputs Inputs Template

At this point, you've created a new virtual machine and attached a data disk. Next you connect to the VM.

Connect to your VM

- In the Azure portal, select **Connect** on the virtual machine properties page.

afsvm101918 Virtual machine

Search (Ctrl+ /) Connect Start Restart Stop Capture Delete Refresh

Overview Resource group (change) afsresgroup101918 Computer name afsvm101918

- On the **Connect to virtual machine** page, keep the default options to connect by **IP address** over port 3389. Select **Download RDP file**.

Connect to virtual machine

afsvm101918

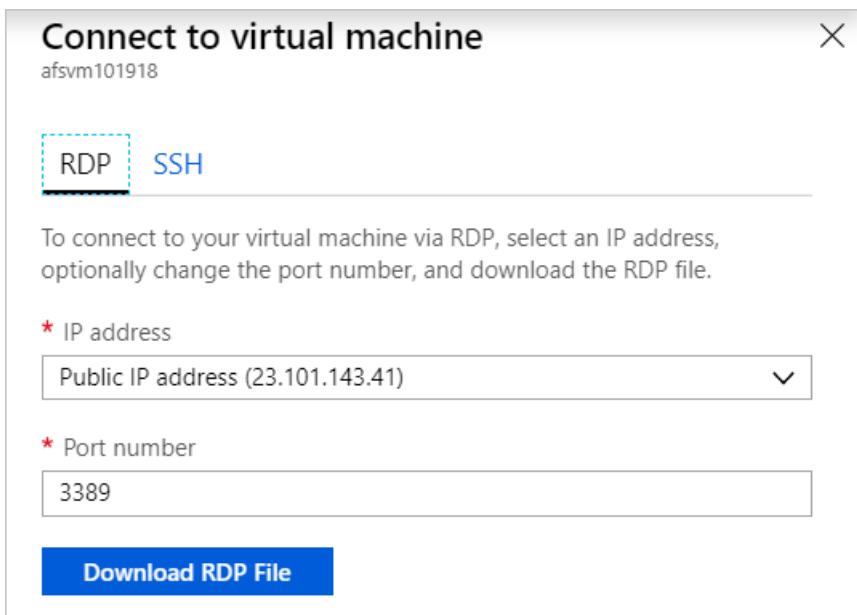
RDP **SSH**

To connect to your virtual machine via RDP, select an IP address, optionally change the port number, and download the RDP file.

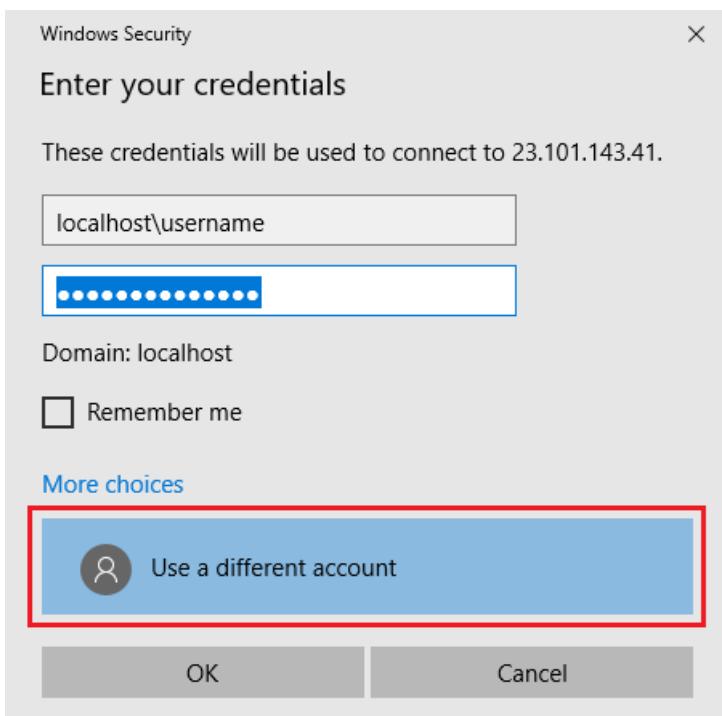
* IP address
Public IP address (23.101.143.41)

* Port number
3389

Download RDP File



3. Open the downloaded RDP file and select **Connect** when prompted.
4. In the **Windows Security** window, select **More choices** and then **Use a different account**. Type the username as *localhost\username*, enter the password you created for the virtual machine, and then select **OK**.



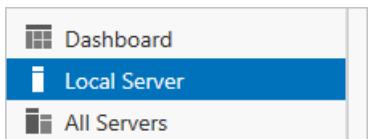
5. You might receive a certificate warning during the sign-in process. Select **Yes** or **Continue** to create the connection.

Prepare the Windows server

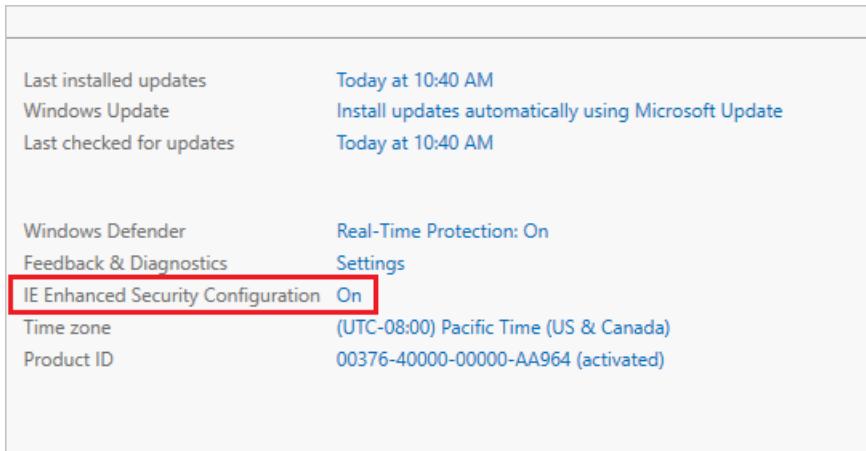
For the Windows Server 2016 Datacenter server, disable Internet Explorer Enhanced Security Configuration. This step is required only for initial server registration. You can re-enable it after the server has been registered.

In the Windows Server 2016 Datacenter VM, Server Manager opens automatically. If Server Manager doesn't open by default, search for it in Start Menu.

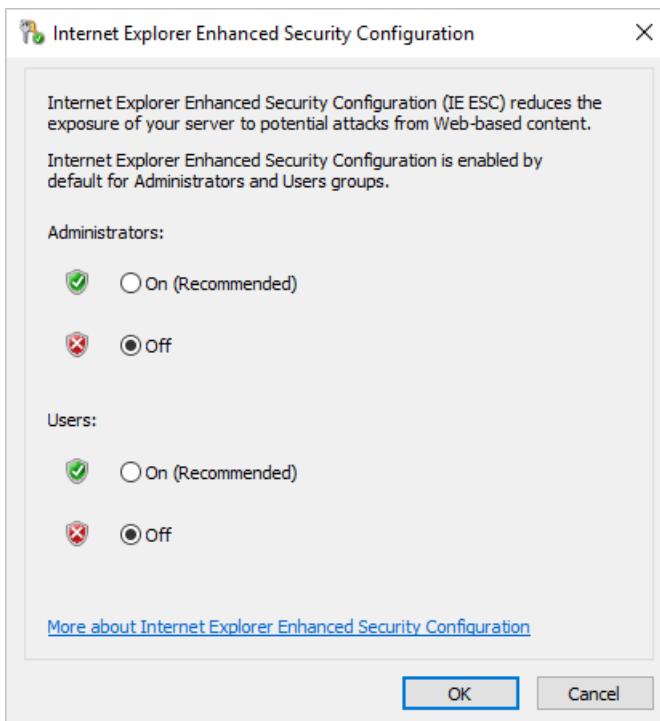
1. In **Server Manager**, select **Local Server**.



2. On the **Properties** pane, select the link for **IE Enhanced Security Configuration**.



3. In the **Internet Explorer Enhanced Security Configuration** dialog box, select **Off** for **Administrators** and **Users**.



Now you can add the data disk to the VM.

Add the data disk

1. While still in the **Windows Server 2016 Datacenter** VM, select **Files and storage services > Volumes > Disks**.

The screenshot shows the 'Disks' section of the Server Manager. The left sidebar has 'Servers', 'Volumes', 'Disks' (which is selected), and 'Storage Pools'. The main area is titled 'DISKS' and shows 'All disks | 3 total'. There is a 'Filter' input field. A table lists three disks:

Number	Virtual Disk	Status	Capacity	Unallocated	Partition	Read Only	Clustered	Subsystem	Bus Type	Name
0		Online	127 GB	2.00 MB	MBR				ATA	Virtual HD
1		Online	7.00 GB	0.00 B	MBR				ATA	Virtual HD
2		Online	1.00 GB	1.00 GB	Unknown				SAS	Msft Virtual Disk

At the bottom, it says 'Last refreshed on 10/15/2018 4:47:56 PM'.

2. Right-click the 1 GB disk named **Msft Virtual Disk** and select **New volume**.
3. Complete the wizard. Use the default settings and make note of the assigned drive letter.
4. Select **Create**.
5. Select **Close**.

At this point, you've brought the disk online and created a volume. Open File Explorer in the Windows Server VM to confirm the presence of the recently added data disk.

6. In File Explorer in the VM, expand **This PC** and open the new drive. It's the F: drive in this example.
7. Right-click and select **New > Folder**. Name the folder *FilesToSync*.
8. Open the **FilesToSync** folder.
9. Right-click and select **New > Text Document**. Name the text file *MyTestFile*.

The screenshot shows File Explorer in a Windows Server VM. The left navigation pane shows 'This PC' expanded, with 'New Volume (F:)' selected. Under 'New Volume (F:)', 'FilesToSync' is selected. The main pane displays a list of files in 'FilesToSync':

Name	Date modified	Type	Size
MyTestFile	10/15/2018 5:05 PM	Text Document	0 KB

10. Close **File Explorer** and **Server Manager**.

Download the Azure PowerShell module

Next, in the Windows Server 2016 Datacenter VM, install the Azure PowerShell module on the server.

1. In the VM, open an elevated PowerShell window.
2. Run the following command:

```
Install-Module -Name Az
```

NOTE

If you have a NuGet version that is older than 2.8.5.201, you're prompted to download and install the latest version of NuGet.

By default, the PowerShell gallery isn't configured as a trusted repository for PowerShellGet. The first time you use the PSGallery, you see the following prompt:

```
Untrusted repository

You are installing the modules from an untrusted repository. If you trust this repository, change its
InstallationPolicy value by running the Set-PSRepository cmdlet.

Are you sure you want to install the modules from 'PSGallery'?
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "N"):
```

3. Answer **Yes** or **Yes to All** to continue with the installation.

The `Az` module is a rollup module for the Azure PowerShell cmdlets. Installing it downloads all the available Azure Resource Manager modules and makes their cmdlets available for use.

At this point, you've set up your environment for the tutorial. You're ready to deploy the Storage Sync Service.

Deploy the service

To deploy Azure File Sync, you first place a **Storage Sync Service** resource into a resource group for your selected subscription. The Storage Sync Service inherits access permissions from its subscription and resource group.

1. In the Azure portal, select **Create a resource** and then search for **Azure File Sync**.
2. In the search results, select **Azure File Sync**.
3. Select **Create** to open the **Deploy Storage Sync** tab.

The screenshot shows the 'Deploy Storage Sync' configuration pane. It includes fields for Name (afssyncservice02), Subscription (Visual Studio Enterprise), Resource group (afsresgroup101918), and Location (East US). The 'Name' field has a green checkmark indicating it's valid.

Name	afssyncservice02
Subscription	Visual Studio Enterprise
Resource group	afsresgroup101918
Location	East US

On the pane that opens, enter the following information:

VALUE	DESCRIPTION
-------	-------------

VALUE	DESCRIPTION
Name	A unique name (per subscription) for the Storage Sync Service. Use <i>afssyncservice02</i> for this tutorial.
Subscription	The Azure subscription you use for this tutorial.
Resource group	The resource group that contains the Storage Sync Service. Use <i>afsresgroup101918</i> for this tutorial.
Location	East US

4. When you're finished, select **Create** to deploy the **Storage Sync Service**.
5. Select the **Notifications** tab > **Go to resource**.

Install the agent

The Azure File Sync agent is a downloadable package that enables Windows Server to be synced with an Azure file share.

1. In the **Windows Server 2016 Datacenter** VM, open **Internet Explorer**.
2. Go to the [Microsoft Download Center](#). Scroll down to the **Azure File Sync Agent** section and select **Download**.

A screenshot of a web page titled "Azure File Sync Agent". Below the title is a message: "Important! Selecting a language below will dynamically change the complete page content to that language." Underneath this, there is a "Language:" dropdown set to "English" and a large orange "Download" button.

3. Select the check box for **StorageSyncAgent_V3_WS2016.EXE** and select **Next**.

A screenshot of a download wizard step titled "Choose the download you want". It shows a list of files with checkboxes:

- File Name
- Microsoft Azure File Sync - License Terms.docx (16 KB)
- StorageSyncAgent_V3_WS2012R2.EXE (51.5 MB)
- StorageSyncAgent_V3_WS2016.EXE (51.5 MB)

4. Select **Allow once** > **Run** > **Open**.
5. If you haven't already done so, close the PowerShell window.
6. Accept the defaults in the **Storage Sync Agent Setup Wizard**.

7. Select **Install**.

8. Select **Finish**.

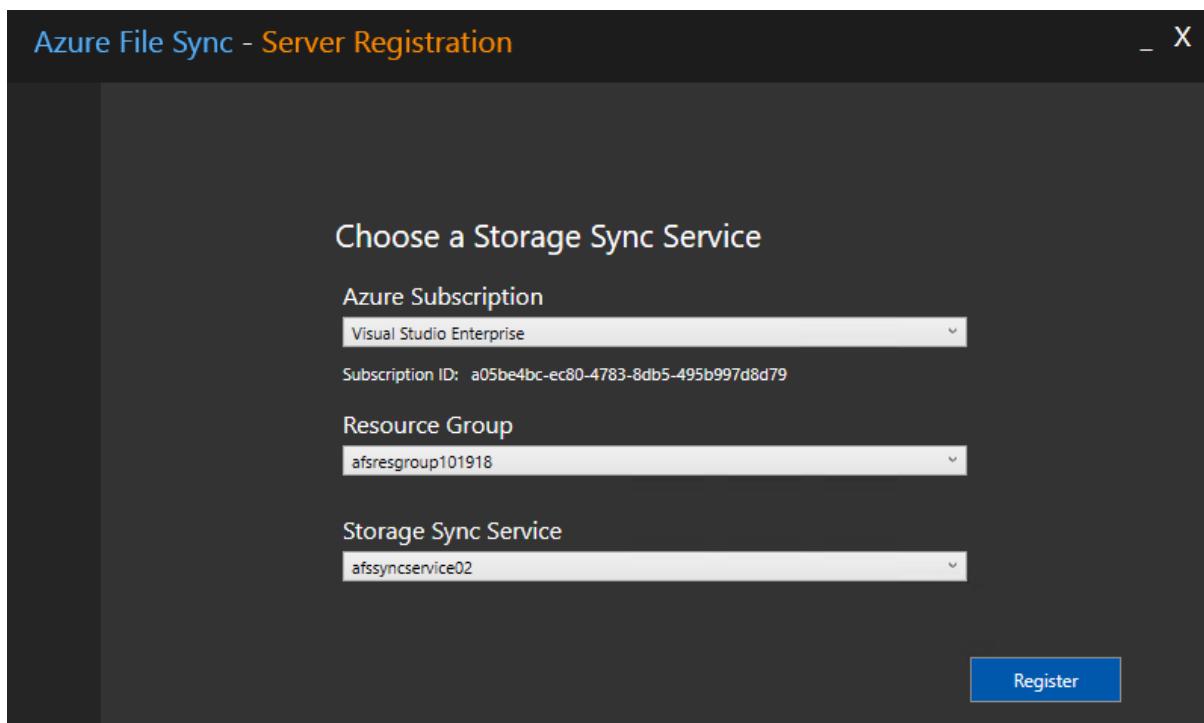
You've deployed the Azure Sync Service and installed the agent on the Windows Server 2016 Datacenter VM. Now you need to register the VM with the Storage Sync Service.

Register Windows Server

Registering your Windows server with a Storage Sync Service establishes a trust relationship between your server (or cluster) and the Storage Sync Service. A server can only be registered to one Storage Sync Service. It can sync with other servers and Azure file shares that are associated with that Storage Sync Service.

The Server Registration UI should open automatically after you install the Azure File Sync agent. If it doesn't, you can open it manually from its file location: `C:\Program Files\Azure\StorageSyncAgent\ServerRegistration.exe`.

1. When the Server Registration UI opens in the VM, select **OK**.
2. Select **Sign-in** to begin.
3. Sign in with your Azure account credentials and select **Sign-in**.
4. Provide the following information:



Value	Description
Azure Subscription	The subscription that contains the Storage Sync Service for this tutorial.
Resource Group	The resource group that contains the Storage Sync Service. Use <i>afsresgroup101918</i> for this tutorial.
Storage Sync Service	The name of the Storage Sync Service. Use <i>afssyncservice02</i> for this tutorial.

5. Select **Register** to complete the server registration.

6. As part of the registration process, you're prompted for an additional sign-in. Sign in and select **Next**.
7. Select **OK**.

Create a sync group

A sync group defines the sync topology for a set of files. A sync group must contain one cloud endpoint, which represents an Azure file share. A sync group also must contain one or more server endpoints. A server endpoint represents a path on a registered server. To create a sync group:

1. In the [Azure portal](#), select **+ Sync group** from the Storage Sync Service. Use *afssyncservice02* for this tutorial.

The screenshot shows the Azure portal interface for a Storage Sync Service named 'afssyncservice02'. The main area displays 'Sync groups' and a 'SYNC GROUP NAME' placeholder. On the left, there's a navigation bar with 'Overview' and 'Activity log' options. At the top right, there's a search bar, a 'Sync group' button (which is highlighted with a red box), and a 'Refresh' button. The overall layout is clean and modern, typical of the Azure UI.

2. Enter the following information to create a sync group with a cloud endpoint:

VALUE	DESCRIPTION
Sync group name	This name must be unique within the Storage Sync Service, but can be any name that is logical for you. Use <i>afssyncgroup</i> for this tutorial.
Subscription	The subscription where you deployed the Storage Sync Service for this tutorial.
Storage account	Choose Select storage account . On the pane that appears, select the storage account that has the Azure file share you created. Use <i>afsstoracct101918</i> for this tutorial.
Azure file share	The name of the Azure file share you created. Use <i>afsfileshare</i> for this tutorial.

3. Select **Create**.

If you select your sync group, you can see that you now have one **cloud endpoint**.

Add a server endpoint

A server endpoint represents a specific location on a registered server. For example, a folder on a server volume. To add a server endpoint:

1. Select the newly created sync group and then select **Add server endpoint**.

Azure File Share

Cloud endpoints

AZURE FILE SHARE	PROVISIONING STATE	RESOURCE GROUP
afsfileshare	✓	afsresgroup101918

Server endpoints

SERVER	HEALTH	FILES NOT SYNCING	SYNC ACTIVITY	PATH	CLOUD TIERING
afsvm101918	✓	0		f:\filestosync	

- On the **Add server endpoint** pane, enter the following information to create a server endpoint:

Value	Description
Registered server	The name of the server you created. Use <i>afsvm101918</i> for this tutorial.
Path	The Windows Server path to the drive you created. Use <i>f:\filestosync</i> in this tutorial.
Cloud Tiering	Leave disabled for this tutorial.
Volume Free Space	Leave blank for this tutorial.

- Select **Create**.

Your files are now in sync across your Azure file share and Windows Server.

afsfileshare

Overview

Access Control (IAM)

Settings

- Access policy
- Properties

Location: afsfileshare

NAME	TYPE	SIZE
.SystemShareInformation	Directory	...
mytestdoc.txt	File	0 B
MyTestFile.txt	File	0 B

Clean up resources

When you're done, you can delete the resource group. Deleting the resource group deletes the storage account, the Azure file share, and any other resources that you deployed inside the resource group.

- In the left menu, select **Resource groups**.
- Right-click the resource group, and then select **Delete resource group**. A window opens and displays a warning about the resources that will be deleted with the resource group.
- Enter the name of the resource group, and then select **Delete**.

Next steps

In this tutorial, you learned the basic steps to extend the storage capacity of a Windows server by using Azure File Sync. For a more thorough look at planning for an Azure File Sync deployment, see:

[Plan for Azure File Sync deployment](#)

Planning for an Azure Files deployment

2/25/2020 • 21 minutes to read • [Edit Online](#)

Azure Files can be deployed in two main ways: by directly mounting the serverless Azure file shares or by caching Azure file shares on-premises using Azure File Sync. Which deployment option you choose changes the things you need to consider as you plan for your deployment.

- **Direct mount of an Azure file share:** Since Azure Files provides SMB access, you can mount Azure file shares on-premises or in the cloud using the standard SMB client available in Windows, macOS, and Linux. Because Azure file shares are serverless, deploying for production scenarios does not require managing a file server or NAS device. This means you don't have to apply software patches or swap out physical disks.
- **Cache Azure file share on-premises with Azure File Sync:** Azure File Sync enables you to centralize your organization's file shares in Azure Files, while keeping the flexibility, performance, and compatibility of an on-premises file server. Azure File Sync transforms an on-premises (or cloud) Windows Server into a quick cache of your Azure file share.

This article primarily addresses deployment considerations for deploying an Azure file share to be directly mounted by an on-premises or cloud client. To plan for an Azure File Sync deployment, see [Planning for an Azure File Sync deployment](#).

Management concepts

Azure file shares are deployed into *storage accounts*, which are top-level objects that represent a shared pool of storage. This pool of storage can be used to deploy multiple file shares, as well as other storage resources such as blob containers, queues, or tables. All storage resources that are deployed into a storage account share the limits that apply to that storage account. To see the current limits for a storage account, see [Azure Files scalability and performance targets](#).

There are two main types of storage accounts you will use for Azure Files deployments:

- **General purpose version 2 (GPv2) storage accounts:** GPv2 storage accounts allow you to deploy Azure file shares on standard/hard disk-based (HDD-based) hardware. In addition to storing Azure file shares, GPv2 storage accounts can store other storage resources such as blob containers, queues, or tables.
- **FileStorage storage accounts:** FileStorage storage accounts allow you to deploy Azure file shares on premium/solid-state disk-based (SSD-based) hardware. FileStorage accounts can only be used to store Azure file shares; no other storage resources (blob containers, queues, tables, etc.) can be deployed in a FileStorage account.

There are several other storage account types you may come across in the Azure portal, PowerShell, or CLI. Two storage account types, BlockBlobStorage and BlobStorage storage accounts, cannot contain Azure file shares. The other two storage account types you may see are general purpose version 1 (GPv1) and classic storage accounts, both of which can contain Azure file shares. Although GPv1 and classic storage accounts may contain Azure file shares, most new features of Azure Files are available only in GPv2 and FileStorage storage accounts. We therefore recommend to only use GPv2 and FileStorage storage accounts for new deployments, and to upgrade GPv1 and classic storage accounts if they already exist in your environment.

When deploying Azure file shares into storage accounts, we recommend:

- Only deploying Azure file shares into storage accounts with other Azure file shares. Although GPv2 storage accounts allow you to have mixed purpose storage accounts, since storage resources such as

Azure file shares and blob containers share the storage account's limits, mixing resources together may make it more difficult to troubleshoot performance issues later on.

- Paying attention to a storage account's IOPS limitations when deploying Azure file shares. Ideally, you would map file shares 1:1 with storage accounts, however this may not always be possible due to various limits and restrictions, both from your organization and from Azure. When it is not possible to have only one file share deployed in one storage account, consider which shares will be highly active and which shares will be less active to ensure that the hottest file shares don't get put in the same storage account together.
- Only deploy GPv2 and FileStorage accounts and upgrade GPv1 and classic storage accounts when you find them in your environment.

Identity

To access an Azure file share, the user of the file share must be authenticated and have authorization to access the share. This is done based on the identity of the user accessing the file share. Azure Files integrates with three main identity providers:

- **Customer-owned Active Directory (preview)**: Azure storage accounts can be domain joined to a customer-owned, Windows Server Active Directory, just like a Windows Server file server or NAS device. Your Active Directory Domain Controller can be deployed on-premises, in an Azure VM, or even as a VM in another cloud provider; Azure Files is agnostic to where your DC is hosted. Once a storage account is domain joined, the end user can mount a file share with the user account they signed into their PC with. AD-based authentication uses the Kerberos authentication protocol.
- **Azure Active Directory Domain Services (Azure AD DS)**: Azure AD DS provides a Microsoft-managed Active Directory Domain Controller that can be used for Azure resources. Domain joining your storage account to Azure AD DS provides similar benefits to domain joining it to a customer-owned Active Directory. This deployment option is most useful for application lift-and-shift scenarios that require AD-based permissions. Since Azure AD DS provides AD-based authentication, this option also uses the Kerberos authentication protocol.
- **Azure storage account key**: Azure file shares may also be mounted with an Azure storage account key. To mount a file share this way, the storage account name is used as the username and the storage account key is used as a password. Using the storage account key to mount the Azure file share is effectively an administrator operation, since the mounted file share will have full permissions to all of the files and folders on the share, even if they have ACLs. When using the storage account key to mount over SMB, the NTLMv2 authentication protocol is used.

For customers migrating from on-premises file servers, or creating new file shares in Azure Files intended to behave like Windows file servers or NAS appliances, domain joining your storage account to **Customer-owned Active Directory** is the recommended option. To learn more about domain joining your storage account to a customer-owned Active Directory, see [Azure Files Active Directory overview](#).

If you intend to use the storage account key to access your Azure file shares, we recommend using service endpoints as described in the [Networking](#) section.

Networking

Azure file shares are accessible from anywhere via the storage account's public endpoint. This means that authenticated requests, such as requests authorized by a user's logon identity, can originate securely from inside or outside of Azure. In many customer environments, an initial mount of the Azure file share on your on-premises workstation will fail, even though mounts from Azure VMs succeed. The reason for this is that many organizations and internet service providers (ISPs) block the port that SMB uses to communicate, port 445.

To unblock access to your Azure file share, you have two main options:

- Unblock port 445 for your organization's on-premises network. Azure file shares may only be externally accessed via the public endpoint using internet safe protocols such as SMB 3.0 and the FileREST API. This is the easiest way to access your Azure file share from on-premises since it doesn't require advanced networking configuration beyond changing your organization's outbound port rules, however, we recommend you remove legacy and deprecated versions of the SMB protocol, namely SMB 1.0. To learn how to do this, see [Securing Windows/Windows Server](#) and [Securing Linux](#).
- Access Azure file shares over an ExpressRoute or VPN connection. When you access your Azure file share via a network tunnel, you are able to mount your Azure file share like an on-premises file share since SMB traffic does not traverse your organizational boundary.

Although from a technical perspective it's considerably easier to mount your Azure file shares via the public endpoint, we expect most customers will opt to mount their Azure file shares over an ExpressRoute or VPN connection. To do this, you will need to configure the following for your environment:

- **Network tunneling using ExpressRoute, Site-to-Site, or Point-to-Site VPN:** Tunneling into a virtual network allows accessing Azure file shares from on-premises, even if port 445 is blocked.
- **Private endpoints:** Private endpoints give your storage account a dedicated IP address from within the address space of the virtual network. This enables network tunneling without needing to open on-premises networks up to all the of the IP address ranges owned by the Azure storage clusters.
- **DNS forwarding:** Configure your on-premises DNS to resolve the name of your storage account (i.e. `storageaccount.file.core.windows.net` for the public cloud regions) to resolve to the IP address of your private endpoints.

To plan for the networking associated with deploying an Azure file share, see [Azure Files networking considerations](#).

Encryption

Azure Files supports two different types of encryption: encryption in transit, which relates to the encryption used when mounting/accessing the Azure file share, and encryption at rest, which relates to how the data is encrypted when it is stored on disk.

Encryption in transit

By default, all Azure storage accounts have encryption in transit enabled. This means that when you mount a file share over SMB or access it via the FileREST protocol (such as through the Azure portal, PowerShell/CLI, or Azure SDKs), Azure Files will only allow the connection if it is made with SMB 3.0+ with encryption or HTTPS. Clients that do not support SMB 3.0 or clients that support SMB 3.0 but not SMB encryption will not be able to mount the Azure file share if encryption in transit is enabled. For more information about which operating systems support SMB 3.0 with encryption, see our detailed documentation for [Windows](#), [macOS](#), and [Linux](#). All current versions of the PowerShell, CLI, and SDKs support HTTPS.

You can disable encryption in transit for an Azure storage account. When encryption is disabled, Azure Files will also allow SMB 2.1, SMB 3.0 without encryption, and unencrypted FileREST API calls over HTTP. The primary reason to disable encryption in transit is to support a legacy application that must be run on an older operating system, such as Windows Server 2008 R2 or older Linux distribution. Azure Files only allows SMB 2.1 connections within the same Azure region as the Azure file share; an SMB 2.1 client outside of the Azure region of the Azure file share, such as on-premises or in a different Azure region, will not be able to access the file share.

We strongly recommend ensuring encryption of data in-transit is enabled.

For more information about encryption in transit, see [requiring secure transfer in Azure storage](#).

Encryption at rest

All data stored in Azure Files is encrypted at rest using Azure storage service encryption (SSE). Storage service

encryption works similarly to BitLocker on Windows: data is encrypted beneath the file system level. Because data is encrypted beneath the Azure file share's file system, as it's encoded to disk, you don't have to have access to the underlying key on the client to read or write to the Azure file share.

By default, data stored in Azure Files is encrypted with Microsoft-managed keys. With Microsoft-managed keys, Microsoft holds the keys to encrypt/decrypt the data, and is responsible for rotating them on a regular basis. You can also choose to manage your own keys, which gives you control over the rotation process. If you choose to encrypt your file shares with customer-managed keys, Azure Files is authorized to access your keys to fulfill read and write requests from your clients. With customer-managed keys, you can revoke this authorization at any time, but this means that your Azure file share will no longer be accessible via SMB or the FileREST API.

Azure Files uses the same encryption scheme as the other Azure storage services such as Azure Blob storage. To learn more about Azure storage service encryption (SSE), see [Azure storage encryption for data at rest](#).

Storage tiers

Azure Files offers two different tiers of storage, premium and standard, to allow you to tailor your shares to the performance and price requirements of your scenario:

- **Premium file shares:** Premium file shares are backed by solid-state drives (SSDs) and are deployed in the **FileStorage storage account** type. Premium file shares provide consistent high performance and low latency, within single-digit milliseconds for most IO operations, for IO-intensive workloads. This makes them suitable for a wide variety of workloads like databases, web site hosting, and development environments. Premium file shares are only available in a provisioned billing model. For more information on the provisioned billing model for premium file shares, see [Understanding provisioning for premium file shares](#).
- **Standard file shares:** Standard file shares are backed by hard disk drives (HDDs) and are deployed in the **general purpose version 2 (GPv2) storage account** type. Standard file shares provide reliable performance for IO workloads that are less sensitive to performance variability such as general-purpose file shares and dev/test environments. Standard file shares are only available in a pay-as-you-go billing model.

In general, Azure Files features and interoperability with other services are the same between premium file shares and standard file shares, however there are a few important differences:

- **Billing model**
 - Premium file shares are billed using a provisioned billing model, which means you pay for how much storage you provision rather than how much storage you actually ask for.
 - Standard file shares are billed using a pay-as-you-go model, which includes a base cost of storage for how much storage you're actually consuming and then an additional transaction cost based on how you use the share. With standard file shares, your bill will increase if you use (read/write/mount) the Azure file share more.
- **Redundancy options**
 - Premium file shares are only available for locally redundant (LRS) and zone redundant (ZRS) storage.
 - Standard file shares are available for locally redundant, zone redundant, geo-redundant (GRS), and geo-zone redundant (GZRS) storage.
- **Maximum size of file share**
 - Premium file shares can be provisioned for up to 100 TiB without any additional work.
 - By default, standard file shares can span only up to 5 TiB, although the share limit can be increased to 100 TiB by opting into the *large file share* storage account feature flag. Standard file shares may only span up to 100 TiB for locally redundant or zone redundant storage accounts. For more information on increasing
- **Regional availability**
 - Premium file shares are not available in every region, and zone redundant support is available in a smaller subset of regions. To find out if premium file shares are currently available in your region, see

the [products available by region](#) page for Azure. To find out what regions support ZRS, see [Azure Availability Zone support by region](#). To help us prioritize new regions and premium tier features, please fill out this [survey](#).

- Standard file shares are available in every Azure region.
- Azure Kubernetes Service (AKS) supports premium file shares in version 1.13 and later.

Once a file share is created as either a premium or a standard file share, you cannot automatically convert it to the other tier. If you would like to switch to the other tier, you must create a new file share in that tier and manually copy the data from your original share to the new share you created. We recommend using `robocopy` for Windows or `rsync` for macOS and Linux to perform that copy.

Understanding provisioning for premium file shares

Premium file shares are provisioned based on a fixed GiB/IOPS/throughput ratio. For each GiB provisioned, the share will be issued one IOPS and 0.1 MiB/s throughput up to the max limits per share. The minimum allowed provisioning is 100 GiB with min IOPS/throughput.

On a best effort basis, all shares can burst up to three IOPS per GiB of provisioned storage for 60 minutes or longer depending on the size of the share. New shares start with the full burst credit based on the provisioned capacity.

Shares must be provisioned in 1 GiB increments. Minimum size is 100 GiB, next size is 101 GiB and so on.

TIP

Baseline IOPS = 1 * provisioned GiB. (Up to a max of 100,000 IOPS).

Burst Limit = 3 * Baseline IOPS. (Up to a max of 100,000 IOPS).

egress rate = 60 MiB/s + 0.06 * provisioned GiB

ingress rate = 40 MiB/s + 0.04 * provisioned GiB

Provisioned share size is specified by share quota. Share quota can be increased at any time but can be decreased only after 24 hours since the last increase. After waiting for 24 hours without a quota increase, you can decrease the share quota as many times as you like, until you increase it again. IOPS/Throughput scale changes will be effective within a few minutes after the size change.

It is possible to decrease the size of your provisioned share below your used GiB. If you do this, you will not lose data but, you will still be billed for the size used and receive the performance (baseline IOPS, throughput, and burst IOPS) of the provisioned share, not the size used.

The following table illustrates a few examples of these formulae for the provisioned share sizes:

CAPACITY (GiB)	BASELINE IOPS	BURST IOPS	EGRESS (MiB/S)	INGRESS (MiB/S)
100	100	Up to 300	66	44
500	500	Up to 1,500	90	60
1,024	1,024	Up to 3,072	122	81
5,120	5,120	Up to 15,360	368	245
10,240	10,240	Up to 30,720	675	450

Capacity (GiB)	Baseline IOPS	Burst IOPS	Egress (MiB/s)	Ingress (MiB/s)
33,792	33,792	Up to 100,000	2,088	1,392
51,200	51,200	Up to 100,000	3,132	2,088
102,400	100,000	Up to 100,000	6,204	4,136

NOTE

File shares performance is subject to machine network limits, available network bandwidth, IO sizes, parallelism, among many other factors. For example, based on internal testing with 8 KiB read/write IO sizes, a single Windows virtual machine, *Standard F16s_v2*, connected to premium file share over SMB could achieve 20K read IOPS and 15K write IOPS. With 512 MiB read/write IO sizes, the same VM could achieve 1.1 GiB/s egress and 370 MiB/s ingress throughput. To achieve maximum performance scale, spread the load across multiple VMs. Please refer [troubleshooting guide](#) for some common performance issues and workarounds.

Bursting

Premium file shares can burst their IOPS up to a factor of three. Bursting is automated and operates based on a credit system. Bursting works on a best effort basis and the burst limit is not a guarantee, file shares can burst *up to the limit*.

Credits accumulate in a burst bucket whenever traffic for your file share is below baseline IOPS. For example, a 100 GiB share has 100 baseline IOPS. If actual traffic on the share was 40 IOPS for a specific 1-second interval, then the 60 unused IOPS are credited to a burst bucket. These credits will then be used later when operations would exceed the baseline IOPs.

TIP

Size of the burst bucket = Baseline IOPS * 2 * 3600.

Whenever a share exceeds the baseline IOPS and has credits in a burst bucket, it will burst. Shares can continue to burst as long as credits are remaining, though shares smaller than 50 TiB will only stay at the burst limit for up to an hour. Shares larger than 50 TiB can technically exceed this one hour limit, up to two hours but, this is based on the number of burst credits accrued. Each IO beyond baseline IOPS consumes one credit and once all credits are consumed the share would return to baseline IOPS.

Share credits have three states:

- Accruing, when the file share is using less than the baseline IOPS.
- Declining, when the file share is bursting.
- Remaining constant, when there are either no credits or baseline IOPS are in use.

New file shares start with the full number of credits in its burst bucket. Burst credits will not be accrued if the share IOPS fall below baseline IOPS due to throttling by the server.

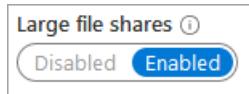
Enable standard file shares to span up to 100 TiB

By default, standard file shares can span only up to 5 TiB, although the share limit can be increased to 100 TiB. To do this, *large file share* feature must be enabled at the storage account-level. Premium storage accounts (*FileStorage* storage accounts) don't have the large file share feature flag as all premium file shares are already enabled for provisioning up to the full 100 TiB capacity.

You can only enable large file shares on locally redundant or zone redundant standard storage accounts. Once you have enabled the large file share feature flag, you can't change the redundancy level to geo-redundant or

geo-zone-redundant storage.

To enable large file shares on an existing storage account, navigate to the **Configuration** view in the storage account's table of contents, and switch the large file share rocker switch to enabled:



You can also enable 100 TiB file shares through the [Set-AzStorageAccount](#) PowerShell cmdlet and the [az storage account update](#) Azure CLI command.

To learn more about how to enable large file shares on new storage accounts, see [creating an Azure file share](#).

Regional availability

Standard file shares with 100 TiB capacity limit are available globally in all Azure regions, except:

- Locally redundant storage: All regions, except for South Africa North, South Africa West, Germany West Central, and Germany North.
- Zone redundant storage: Supported for all regions where Zone redundant storage is supported, except for Japan East, North Europe, South Africa North.
- Geo-redundant/GeoZone redundant storage: Not supported.

Redundancy

To protect the data in your Azure file shares against data loss or corruption, all Azure file shares store multiple copies of each file as they are written. Depending on the requirements of your workload, you can select additional degrees of redundancy. Azure Files currently supports the following data redundancy options:

- **Locally redundant:** Locally redundant storage, often referred to as LRS, means that every file is stored three times within an Azure storage cluster. This protects against loss of data due to hardware faults, such as a bad disk drive.
- **Zone redundant:** Zone redundant storage, often referred to as ZRS, means that every file is stored three times across three distinct Azure storage clusters. The three distinct Azure storage clusters each store the file three times, just like with locally redundant storage, and are physically isolated in different Azure *availability zones*. Availability zones are unique physical locations within an Azure region. Each zone is made up of one or more datacenters equipped with independent power, cooling, and networking. A write to storage is not accepted until it is written to the storage clusters in all three availability zones.
- **Geo-redundant:** Geo-redundant storage, often referred to as GRS, is like locally redundant storage, in that a file is stored three times within an Azure storage cluster in the primary region. All writes are then asynchronously replicated to a Microsoft-defined secondary region. Geo-redundant storage provides 6 copies of your data spread between two Azure regions. In the event of a major disaster such as the permanent loss of an Azure region due to a natural disaster or other similar event, Microsoft will perform a failover so that the secondary in effect becomes the primary, serving all operations. Since the replication between the primary and secondary regions are asynchronous, in the event of a major disaster, data not yet replicated to the secondary region will be lost. You can also perform a manual failover of a geo-redundant storage account.
- **Geo-zone redundant:** Geo-zone redundant storage, often referred to as GZRS, is like zone redundant storage, in that a file is stored nine times across 3 distinct storage clusters in the primary region. All writes are then asynchronously replicated to a Microsoft-defined secondary region. The failover process for geo-zone-redundant storage works the same as it does for geo-redundant storage.

Standard Azure file shares support all four redundancy types, while premium Azure file shares only support locally redundant and zone redundant storage.

General purpose version 2 (GPv2) storage accounts provide two additional redundancy options that are not supported by Azure Files: read accessible geo-redundant storage, often referred to as RA-GRS, and read

accessible geo-zone-redundant storage, often referred to as RA-GZRS. You can provision Azure file shares in storage accounts with these options set, however Azure Files does not support reading from the secondary region. Azure file shares deployed into read-accessible geo- or geo-zone redundant storage accounts will be billed as geo-redundant or geo-zone-redundant storage, respectively.

Migration

In many cases, you will not be establishing a net new file share for your organization, but instead migrating an existing file share from an on-premises file server or NAS device to Azure Files. There are many tools, provided both by Microsoft and 3rd parties, to do a migration to a file share, but they can roughly be divided into two categories:

- **Tools which maintain file system attributes such as ACLs and timestamps:**
 - **Azure File Sync:** Azure File Sync can be used as a method to ingest data into an Azure file share, even when the desired end deployment isn't to maintain an on-premises presence. Azure File Sync can be installed in place on existing Windows Server 2012 R2, Windows Server 2016, and Windows Server 2019 deployments. An advantage to using Azure File Sync as an ingest mechanism is that end users can continue to use the existing file share in place; cut-over to the Azure file share can occur after all of the data is finished uploading in the background.
 - **Robocopy:** Robocopy is a well-known copy tool that ships with Windows and Windows Server. Robocopy may be used to transfer data into Azure Files by mounting the file share locally, and then using the mounted location as the destination in the Robocopy command.
- **Tools which do not maintain file system attributes:**
 - **Data Box:** Data Box provides an offline data transfer mechanism to physical ship data into Azure. This method is designed to increase throughput and save bandwidth, but does not currently support file system attributes like timestamps and ACLs.
 - **AzCopy:** AzCopy is a command-line utility designed for copying data to and from Azure Files, as well as Azure Blob storage, using simple commands with optimal performance.

Next steps

- [Planning for an Azure File Sync Deployment](#)
- [Deploying Azure Files](#)
- [Deploying Azure File Sync](#)

Planning for an Azure File Sync deployment

2/25/2020 • 36 minutes to read • [Edit Online](#)

Azure Files can be deployed in two main ways: by directly mounting the serverless Azure file shares or by caching Azure file shares on-premises using Azure File Sync. Which deployment option you choose changes the things you need to consider as you plan for your deployment.

- **Direct mount of an Azure file share:** Since Azure Files provides SMB access, you can mount Azure file shares on-premises or in the cloud using the standard SMB client available in Windows, macOS, and Linux. Because Azure file shares are serverless, deploying for production scenarios does not require managing a file server or NAS device. This means you don't have to apply software patches or swap out physical disks.
- **Cache Azure file share on-premises with Azure File Sync:** Azure File Sync enables you to centralize your organization's file shares in Azure Files, while keeping the flexibility, performance, and compatibility of an on-premises file server. Azure File Sync transforms an on-premises (or cloud) Windows Server into a quick cache of your Azure file share.

This article primarily addresses deployment considerations for deploying Azure File Sync. To plan for a deployment of Azure file shares to be directly mounted by an on-premises or cloud client, see [Planning for an Azure Files deployment](#).

Management concepts

An Azure File Sync deployment has three fundamental management objects:

- **Azure file share:** An Azure file share is a serverless cloud file share, which provides the *cloud endpoint* of an Azure File Sync sync relationship. Files in an Azure file share can be accessed directly with SMB or the FileREST protocol, although we encourage you to primarily access the files through the Windows Server cache when the Azure file share is being used with Azure File Sync. This is because Azure Files today lacks an efficient change detection mechanism like Windows Server has, so changes to the Azure file share directly will take time to propagate back to the server endpoints.
- **Server endpoint:** The path on the Windows Server that is being synced to an Azure file share. This can be a specific folder on a volume or the root of the volume. Multiple server endpoints can exist on the same volume if their namespaces do not overlap.
- **Sync group:** The object that defines the sync relationship between a **cloud endpoint**, or Azure file share, and a server endpoint. Endpoints within a sync group are kept in sync with each other. If for example, you have two distinct sets of files that you want to manage with Azure File Sync, you would create two sync groups and add different endpoints to each sync group.

Azure file share management concepts

Azure file shares are deployed into *storage accounts*, which are top-level objects that represent a shared pool of storage. This pool of storage can be used to deploy multiple file shares, as well as other storage resources such as blob containers, queues, or tables. All storage resources that are deployed into a storage account share the limits that apply to that storage account. To see the current limits for a storage account, see [Azure Files scalability and performance targets](#).

There are two main types of storage accounts you will use for Azure Files deployments:

- **General purpose version 2 (GPv2) storage accounts:** GPv2 storage accounts allow you to deploy Azure file shares on standard/hard disk-based (HDD-based) hardware. In addition to storing Azure file shares,

GPv2 storage accounts can store other storage resources such as blob containers, queues, or tables.

- **FileStorage storage accounts:** FileStorage storage accounts allow you to deploy Azure file shares on premium/solid-state disk-based (SSD-based) hardware. FileStorage accounts can only be used to store Azure file shares; no other storage resources (blob containers, queues, tables, etc.) can be deployed in a FileStorage account.

There are several other storage account types you may come across in the Azure portal, PowerShell, or CLI. Two storage account types, BlockBlobStorage and BlobStorage storage accounts, cannot contain Azure file shares. The other two storage account types you may see are general purpose version 1 (GPv1) and classic storage accounts, both of which can contain Azure file shares. Although GPv1 and classic storage accounts may contain Azure file shares, most new features of Azure Files are available only in GPv2 and FileStorage storage accounts. We therefore recommend to only use GPv2 and FileStorage storage accounts for new deployments, and to upgrade GPv1 and classic storage accounts if they already exist in your environment.

Azure File Sync management concepts

Sync groups are deployed into **Storage Sync Services**, which are top-level objects that register servers for use with Azure File Sync and contain the sync group relationships. The Storage Sync Service resource is a peer of the storage account resource, and can similarly be deployed to Azure resource groups. A Storage Sync Service can create sync groups that contain Azure file shares across multiple storage accounts and multiple registered Windows Servers.

Before you can create a sync group in a Storage Sync Service, you must first register a Windows Server with the Storage Sync Service. This creates a **registered server** object, which represents a trust relationship between your server or cluster and the Storage Sync Service. To register a Storage Sync Service, you must first install the Azure File Sync agent on the server. An individual server or cluster can be registered with only one Storage Sync Service at a time.

A sync group contains one cloud endpoint, or Azure file share, and at least one server endpoint. The server endpoint object contains the settings that configure the **cloud tiering** capability, which provides the caching capability of Azure File Sync. In order to sync with an Azure file share, the storage account containing the Azure file share must be in the same Azure region as the Storage Sync Service.

Management guidance

When deploying Azure File Sync, we recommend:

- Deploying Azure file shares 1:1 with Windows file shares. The server endpoint object gives you a great degree of flexibility on how you set up the sync topology on the server-side of the sync relationship. To simplify management, make the path of the server endpoint match the path of the Windows file share.
- Use as few Storage Sync Services as possible. This will simplify management when you have sync groups that contain multiple server endpoints, since a Windows Server can only be registered to one Storage Sync Service at a time.
- Paying attention to a storage account's IOPS limitations when deploying Azure file shares. Ideally, you would map file shares 1:1 with storage accounts, however this may not always be possible due to various limits and restrictions, both from your organization and from Azure. When it is not possible to have only one file share deployed in one storage account, consider which shares will be highly active and which shares will be less active to ensure that the hottest file shares don't get put in the same storage account together.

Windows file server considerations

To enable the sync capability on Windows Server, you must install the Azure File Sync downloadable agent. The Azure File Sync agent provides two main components: `FileSyncSvc.exe`, the background Windows service that is responsible for monitoring changes on the server endpoints and initiating sync sessions, and

`StorageSync.sys`, a file system filter which enables cloud tiering and fast disaster recovery.

Operating system requirements

Azure File Sync is supported with the following versions of Windows Server:

VERSION	SUPPORTED SKUS	SUPPORTED DEPLOYMENT OPTIONS
Windows Server 2019	Datacenter, Standard, and IoT	Full and Core
Windows Server 2016	Datacenter, Standard, and Storage Server	Full and Core
Windows Server 2012 R2	Datacenter, Standard, and Storage Server	Full and Core

Future versions of Windows Server will be added as they are released.

IMPORTANT

We recommend keeping all servers that you use with Azure File Sync up to date with the latest updates from Windows Update.

Minimum system resources

Azure File Sync requires a server, either physical or virtual, with at least one CPU and a minimum of 2 GiB of memory.

IMPORTANT

If the server is running in a virtual machine with dynamic memory enabled, the VM should be configured with a minimum of 2048 MiB of memory.

For most production workloads, we do not recommend configuring an Azure File Sync sync server with only the minimum requirements. See [Recommended system resources](#) for more information.

Recommended system resources

Just like any server feature or application, the system resource requirements for Azure File Sync are determined by the scale of the deployment; larger deployments on a server require greater system resources. For Azure File Sync, scale is determined by the number of objects across the server endpoints and the churn on the dataset. A single server can have server endpoints in multiple sync groups and the number of objects listed in the following table accounts for the full namespace that a server is attached to.

For example, server endpoint A with 10 million objects + server endpoint B with 10 million objects = 20 million objects. For that example deployment, we would recommend 8 CPUs, 16 GiB of memory for steady state, and (if possible) 48 GiB of memory for the initial migration.

Namespace data is stored in memory for performance reasons. Because of that, bigger namespaces require more memory to maintain good performance, and more churn requires more CPU to process.

In the following table, we have provided both the size of the namespace as well as a conversion to capacity for typical general purpose file shares, where the average file size is 512 KiB. If your file sizes are smaller, consider adding additional memory for the same amount of capacity. Base your memory configuration on the size of the namespace.

NAMESPACE SIZE - FILES & DIRECTORIES (MILLIONS)	TYPICAL CAPACITY (TIB)	CPU CORES	RECOMMENDED MEMORY (GIB)
3	1.4	2	8 (initial sync)/ 2 (typical churn)
5	2.3	2	16 (initial sync)/ 4 (typical churn)
10	4.7	4	32 (initial sync)/ 8 (typical churn)
30	14.0	8	48 (initial sync)/ 16 (typical churn)
50	23.3	16	64 (initial sync)/ 32 (typical churn)
100*	46.6	32	128 (initial sync)/ 32 (typical churn)

*Syncing more than 100 million files & directories is not recommended at this time. This is a soft limit based on our tested thresholds. For more information, see [Azure Files scalability and performance targets](#).

TIP

Initial synchronization of a namespace is an intensive operation and we recommend allocating more memory until initial synchronization is complete. This isn't required but, may speed up initial sync.

Typical churn is 0.5% of the namespace changing per day. For higher levels of churn, consider adding more CPU.

- A locally attached volume formatted with the NTFS file system.

Evaluation cmdlet

Before deploying Azure File Sync, you should evaluate whether it is compatible with your system using the Azure File Sync evaluation cmdlet. This cmdlet checks for potential issues with your file system and dataset, such as unsupported characters or an unsupported operating system version. Its checks cover most but not all of the features mentioned below; we recommend you read through the rest of this section carefully to ensure your deployment goes smoothly.

The evaluation cmdlet can be installed by installing the Az PowerShell module, which can be installed by following the instructions here: [Install and configure Azure PowerShell](#).

Usage

You can invoke the evaluation tool in a few different ways: you can perform the system checks, the dataset checks, or both. To perform both the system and dataset checks:

```
Invoke-AzStorageSyncCompatibilityCheck -Path <path>
```

To test only your dataset:

```
Invoke-AzStorageSyncCompatibilityCheck -Path <path> -SkipSystemChecks
```

To test system requirements only:

```
Invoke-AzStorageSyncCompatibilityCheck -ComputerName <computer name>
```

To display the results in CSV:

```
$errors = Invoke-AzStorageSyncCompatibilityCheck [...]  
$errors | Select-Object -Property Type, Path, Level, Description | Export-Csv -Path <csv path>
```

File system compatibility

Azure File Sync is only supported on directly attached, NTFS volumes. Direct attached storage, or DAS, on Windows Server means that the Windows Server operating system owns the file system. DAS can be provided through physically attaching disks to the file server, attaching virtual disks to a file server VM (such as a VM hosted by Hyper-V), or even through iSCSI.

Only NTFS volumes are supported; ReFS, FAT, FAT32, and other file systems are not supported.

The following table shows the interop state of NTFS file system features:

FEATURE	SUPPORT STATUS	NOTES
Access control lists (ACLs)	Fully supported	Windows-style discretionary access control lists are preserved by Azure File Sync, and are enforced by Windows Server on server endpoints. ACLs can also be enforced when directly mounting the Azure file share, however this requires additional configuration. See the Identity section for more information.
Hard links	Skipped	
Symbolic links	Skipped	
Mount points	Partially supported	Mount points might be the root of a server endpoint, but they are skipped if they are contained in a server endpoint's namespace.
Junctions	Skipped	For example, Distributed File System DfrsrPrivate and DFSRoots folders.
Reparse points	Skipped	
NTFS compression	Fully supported	
Sparse files	Fully supported	Sparse files sync (are not blocked), but they sync to the cloud as a full file. If the file contents change in the cloud (or on another server), the file is no longer sparse when the change is downloaded.

FEATURE	SUPPORT STATUS	NOTES
Alternate Data Streams (ADS)	Preserved, but not synced	For example, classification tags created by the File Classification Infrastructure are not synced. Existing classification tags on files on each of the server endpoints are left untouched.

Azure File Sync will also skip certain temporary files and system folders:

FILE/FOLDER	NOTE
pagefile.sys	File specific to system
Desktop.ini	File specific to system
thumbs.db	Temporary file for thumbnails
ehthumbs.db	Temporary file for media thumbnails
~\$.*	Office temporary file
*.tmp	Temporary file
*.laccdb	Access DB locking file
635D02A9D91C401B97884B82B3BCDAEA.*	Internal Sync file
\System Volume Information	Folder specific to volume
\$RECYCLE.BIN	Folder
\SyncShareState	Folder for Sync

Failover Clustering

Windows Server Failover Clustering is supported by Azure File Sync for the "File Server for general use" deployment option. Failover Clustering is not supported on "Scale-Out File Server for application data" (SOFS) or on Clustered Shared Volumes (CSVs).

NOTE

The Azure File Sync agent must be installed on every node in a Failover Cluster for sync to work correctly.

Data Deduplication

Windows Server 2016 and Windows Server 2019

Data Deduplication is supported on volumes with cloud tiering enabled on Windows Server 2016 and Windows Server 2019. Enabling Data Deduplication on a volume with cloud tiering enabled lets you cache more files on-premises without provisioning more storage.

When Data Deduplication is enabled on a volume with cloud tiering enabled, Dedup optimized files within the server endpoint location will be tiered similar to a normal file based on the cloud tiering policy settings. Once the Dedup optimized files have been tiered, the Data Deduplication garbage collection job will run automatically to reclaim disk space by removing unnecessary chunks that are no longer referenced by other files on the

volume.

Note the volume savings only apply to the server; your data in the Azure file share will not be deduped.

NOTE

To support Data Deduplication on volumes with cloud tiering enabled on Windows Server 2019, Windows update [KB4520062](#) must be installed and Azure File Sync agent version 9.0.0.0 or newer is required.

Windows Server 2012 R2

Azure File Sync does not support Data Deduplication and cloud tiering on the same volume on Windows Server 2012 R2. If Data Deduplication is enabled on a volume, cloud tiering must be disabled.

Notes

- If Data Deduplication is installed prior to installing the Azure File Sync agent, a restart is required to support Data Deduplication and cloud tiering on the same volume.
- If Data Deduplication is enabled on a volume after cloud tiering is enabled, the initial Deduplication optimization job will optimize files on the volume that are not already tiered and will have the following impact on cloud tiering:
 - Free space policy will continue to tier files as per the free space on the volume by using the heatmap.
 - Date policy will skip tiering of files that may have been otherwise eligible for tiering due to the Deduplication optimization job accessing the files.
- For ongoing Deduplication optimization jobs, cloud tiering with date policy will get delayed by the Data Deduplication [MinimumFileAgeDays](#) setting, if the file is not already tiered.
 - Example: If the MinimumFileAgeDays setting is seven days and cloud tiering date policy is 30 days, the date policy will tier files after 37 days.
 - Note: Once a file is tiered by Azure File Sync, the Deduplication optimization job will skip the file.
- If a server running Windows Server 2012 R2 with the Azure File Sync agent installed is upgraded to Windows Server 2016 or Windows Server 2019, the following steps must be performed to support Data Deduplication and cloud tiering on the same volume:
 - Uninstall the Azure File Sync agent for Windows Server 2012 R2 and restart the server.
 - Download the Azure File Sync agent for the new server operating system version (Windows Server 2016 or Windows Server 2019).
 - Install the Azure File Sync agent and restart the server.

Note: The Azure File Sync configuration settings on the server are retained when the agent is uninstalled and reinstalled.

Distributed File System (DFS)

Azure File Sync supports interop with DFS Namespaces (DFS-N) and DFS Replication (DFS-R).

DFS Namespaces (DFS-N): Azure File Sync is fully supported on DFS-N servers. You can install the Azure File Sync agent on one or more DFS-N members to sync data between the server endpoints and the cloud endpoint. For more information, see [DFS Namespaces overview](#).

DFS Replication (DFS-R): Since DFS-R and Azure File Sync are both replication solutions, in most cases, we recommend replacing DFS-R with Azure File Sync. There are however several scenarios where you would want to use DFS-R and Azure File Sync together:

- You are migrating from a DFS-R deployment to an Azure File Sync deployment. For more information, see [Migrate a DFS Replication \(DFS-R\) deployment to Azure File Sync](#).

- Not every on-premises server that needs a copy of your file data can be connected directly to the internet.
- Branch servers consolidate data onto a single hub server, for which you would like to use Azure File Sync.

For Azure File Sync and DFS-R to work side by side:

1. Azure File Sync cloud tiering must be disabled on volumes with DFS-R replicated folders.
2. Server endpoints should not be configured on DFS-R read-only replication folders.

For more information, see [DFS Replication overview](#).

Sysprep

Using sysprep on a server that has the Azure File Sync agent installed is not supported and can lead to unexpected results. Agent installation and server registration should occur after deploying the server image and completing sysprep mini-setup.

Windows Search

If cloud tiering is enabled on a server endpoint, files that are tiered are skipped and not indexed by Windows Search. Non-tiered files are indexed properly.

Other Hierarchical Storage Management (HSM) solutions

No other HSM solutions should be used with Azure File Sync.

Identity

Azure File Sync works with your standard AD-based identity without any special setup beyond setting up sync. When you are using Azure File Sync, the general expectation is that most accesses go through the Azure File Sync caching servers, rather than through the Azure file share. Since the server endpoints are located on Windows Server, and Windows Server has supported AD and Windows-style ACLs for a very long time, nothing is needed beyond ensuring the Windows file servers registered with the Storage Sync Service are domain joined. Azure File Sync will store ACLs on the files in the Azure file share, and will replicate them to all server endpoints.

Even though changes made directly to the Azure file share will take longer to sync to the server endpoints in the sync group, you may also want to ensure that you can enforce your AD permissions on your file share directly in the cloud as well. To do this, you must domain join your storage account to your on-premises AD, just like how your Windows file servers are domain joined. To learn more about domain joining your storage account to a customer-owned Active Directory, see [Azure Files Active Directory overview](#).

IMPORTANT

Domain joining your storage account to Active Directory is not required to successfully deploy Azure File Sync. This is a strictly optional step that allows the Azure file share to enforce on-premises ACLs when users mount the Azure file share directly.

Networking

The Azure File Sync agent communicates with your Storage Sync Service and Azure file share using the Azure File Sync REST protocol and the FileREST protocol, both of which always use HTTPS over port 443. SMB is never used to upload or download data between your Windows Server and the Azure file share. Because most organizations allow HTTPS traffic over port 443, as a requirement for visiting most websites, special networking configuration is usually not required to deploy Azure File Sync.

Based on your organization's policy or unique regulatory requirements, you may require more restrictive communication with Azure, and therefore Azure File Sync provides several mechanisms for you to configure networking. Based on your requirements, you can:

- Tunnel sync and file upload/download traffic over your ExpressRoute or Azure VPN.
- Make use of Azure Files and Azure Networking features such as service endpoints and private endpoints.
- Configure Azure File Sync to support your proxy in your environment.
- Throttle network activity from Azure File Sync.

To learn more about configuring the networking functionality of Azure File Sync, see:

- [Azure File Sync proxy and firewall settings](#)
- [Ensuring Azure File Sync is a good neighbor in your datacenter](#)

Encryption

When using Azure File Sync, there are three different layers of encryption to consider: encryption on the at-rest storage of Windows Server, encryption in transit between the Azure File Sync agent and Azure, and encryption at rest of your data in the Azure file share.

Windows Server encryption at rest

There are two strategies for encrypting data on Windows Server that work generally with Azure File Sync: encryption beneath the file system such that the file system and all of the data written to it is encrypted, and encryption within the file format itself. These methods are not mutually exclusive; they can be used together if desired since the purpose of encryption is different.

To provide encryption beneath the file system, Windows Server provides BitLocker inbox. BitLocker is fully transparent to Azure File Sync. The primary reason to use an encryption mechanism like BitLocker is to prevent physical exfiltration of data from your on-premises datacenter by someone stealing the disks and to prevent sideloading an unauthorized OS to perform unauthorized reads/writes to your data. To learn more about BitLocker, see [BitLocker overview](#).

Third party products which work similarly to BitLocker, in that they sit beneath the NTFS volume, should similarly work fully transparently with Azure File Sync.

The other main method for encrypting data is to encrypt the file's data stream when the application saves the file. Some applications may do this natively, however this is usually not the case. An example of a method for encrypting the file's data stream is Azure Information Protection (AIP)/Azure Rights Management Services (Azure RMS)/Active Directory RMS. The primary reason to use an encryption mechanism like AIP/RMS is to prevent data exfiltration of data from your file share by people copying it to alternate locations, like to a flash drive, or emailing it to an unauthorized person. When a file's data stream is encrypted as part of the file format, this file will continue to be encrypted on the Azure file share.

Azure File Sync does not interoperate with NTFS Encrypted File System (NTFS EFS) or third party encryption solutions that sit above the file system but below the file's data stream.

Encryption in transit

Azure File Sync agent communicates with your Storage Sync Service and Azure file share using the Azure File Sync REST protocol and the FileREST protocol, both of which always use HTTPS over port 443. Azure File Sync does not send unencrypted requests over HTTP.

Azure storage accounts contain a switch for requiring encryption in transit, which is enabled by default. Even if the switch at the storage account level is disabled, meaning that unencrypted connections to your Azure file shares are possible, Azure File Sync will still only use encrypted channels to access your file share.

The primary reason to disable encryption in transit for the storage account is to support a legacy application that must be run on an older operating system, such as Windows Server 2008 R2 or older Linux distribution, talking to an Azure file share directly. If the legacy application talks to the Windows Server cache of the file share, toggling this setting will have no effect.

We strongly recommend ensuring encryption of data in-transit is enabled.

For more information about encryption in transit, see [requiring secure transfer in Azure storage](#).

Azure file share encryption at rest

All data stored in Azure Files is encrypted at rest using Azure storage service encryption (SSE). Storage service encryption works similarly to BitLocker on Windows: data is encrypted beneath the file system level. Because data is encrypted beneath the Azure file share's file system, as it's encoded to disk, you don't have to have access to the underlying key on the client to read or write to the Azure file share.

By default, data stored in Azure Files is encrypted with Microsoft-managed keys. With Microsoft-managed keys, Microsoft holds the keys to encrypt/decrypt the data, and is responsible for rotating them on a regular basis.

You can also choose to manage your own keys, which gives you control over the rotation process. If you choose to encrypt your file shares with customer-managed keys, Azure Files is authorized to access your keys to fulfill read and write requests from your clients. With customer-managed keys, you can revoke this authorization at any time, but this means that your Azure file share will no longer be accessible via SMB or the FileREST API.

Azure Files uses the same encryption scheme as the other Azure storage services such as Azure Blob storage. To learn more about Azure storage service encryption (SSE), see [Azure storage encryption for data at rest](#).

Storage tiers

Azure Files offers two different tiers of storage, premium and standard, to allow you to tailor your shares to the performance and price requirements of your scenario:

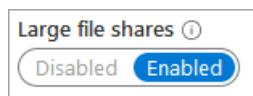
- **Premium file shares:** Premium file shares are backed by solid-state drives (SSDs) and are deployed in the **FileStorage storage account** type. Premium file shares provide consistent high performance and low latency, within single-digit milliseconds for most IO operations, for IO-intensive workloads. This makes them suitable for a wide variety of workloads like databases, web site hosting, and development environments. Premium file shares are only available in a provisioned billing model. For more information on the provisioned billing model for premium file shares, see [Understanding provisioning for premium file shares](#).
- **Standard file shares:** Standard file shares are backed by hard disk drives (HDDs) and are deployed in the **general purpose version 2 (GPv2) storage account** type. Standard file shares provide reliable performance for IO workloads that are less sensitive to performance variability such as general-purpose file shares and dev/test environments. Standard file shares are only available in a pay-as-you-go billing model.

Enable standard file shares to span up to 100 TiB

By default, standard file shares can span only up to 5 TiB, although the share limit can be increased to 100 TiB. To do this, *large file share* feature must be enabled at the storage account-level. Premium storage accounts (*FileStorage* storage accounts) don't have the large file share feature flag as all premium file shares are already enabled for provisioning up to the full 100 TiB capacity.

You can only enable large file shares on locally redundant or zone redundant standard storage accounts. Once you have enabled the large file share feature flag, you can't change the redundancy level to geo-redundant or geo-zone-redundant storage.

To enable large file shares on an existing storage account, navigate to the **Configuration** view in the storage account's table of contents, and switch the large file share rocker switch to enabled:



You can also enable 100 TiB file shares through the [Set-AzStorageAccount](#) PowerShell cmdlet and the [az storage account update](#) Azure CLI command.

To learn more about how to enable large file shares on new storage accounts, see [creating an Azure file share](#).

Regional availability

Standard file shares with 100 TiB capacity limit are available globally in all Azure regions, except:

- Locally redundant storage: All regions, except for South Africa North, South Africa West, Germany West Central, and Germany North.
- Zone redundant storage: Supported for all regions where Zone redundant storage is supported, except for Japan East, North Europe, South Africa North.
- Geo-redundant/GeoZone redundant storage: Not supported.

Azure file sync region availability

Azure File Sync is available in the following regions:

AZURE CLOUD	GEOGRAPHIC REGION	AZURE REGION	REGION CODE
Public	Asia	East Asia	eastasia
Public	Asia	Southeast Asia	southeastasia
Public	Australia	Australia East	australiaeast
Public	Australia	Australia Southeast	australiasoutheast
Public	Brazil	Brazil South	brazilsouth
Public	Canada	Canada Central	canadacentral
Public	Canada	Canada East	canadaeast
Public	Europe	North Europe	northeurope
Public	Europe	West Europe	westeurope
Public	France	France Central	francecentral
Public	France	France South*	francesouth
Public	India	Central India	centralindia
Public	India	South India	southindia
Public	Japan	Japan East	japaneast
Public	Japan	Japan West	japanwest
Public	Korea	Korea Central	koreacentral
Public	Korea	Korea South	koreasouth
Public	South Africa	South Africa North	southafricanorth

AZURE CLOUD	GEOGRAPHIC REGION	AZURE REGION	REGION CODE
Public	South Africa	South Africa West*	southafricawest
Public	UAE	UAE Central*	uaecentral
Public	UAE	UAE North	uaenorth
Public	UK	UK South	uksouth
Public	UK	UK West	ukwest
Public	US	Central US	centralus
Public	US	East US	eastus
Public	US	East US 2	eastus2
Public	US	North Central US	northcentralus
Public	US	South Central US	southcentralus
Public	US	West Central US	westcentralus
Public	US	West US	westus
Public	US	West US 2	westus2
US Gov	US	US Gov Arizona	usgovarizona
US Gov	US	US Gov Texas	usgovtexas
US Gov	US	US Gov Virginia	usgovvirginia

Azure File Sync supports syncing only with an Azure file share that's in the same region as the Storage Sync Service.

For the regions marked with asterisks, you must contact Azure Support to request access to Azure Storage in those regions. The process is outlined in [this document](#).

Redundancy

To protect the data in your Azure file shares against data loss or corruption, all Azure file shares store multiple copies of each file as they are written. Depending on the requirements of your workload, you can select additional degrees of redundancy. Azure Files currently supports the following data redundancy options:

- **Locally redundant:** Locally redundant storage, often referred to as LRS, means that every file is stored three times within an Azure storage cluster. This protects against loss of data due to hardware faults, such as a bad disk drive.
- **Zone redundant:** Zone redundant storage, often referred to as ZRS, means that every file is stored three times across three distinct Azure storage clusters. The three distinct Azure storage clusters each store the file three times, just like with locally redundant storage, and are physically isolated in different Azure *availability zones*.

zones. Availability zones are unique physical locations within an Azure region. Each zone is made up of one or more datacenters equipped with independent power, cooling, and networking. A write to storage is not accepted until it is written to the storage clusters in all three availability zones.

- **Geo-redundant:** Geo-redundant storage, often referred to as GRS, is like locally redundant storage, in that a file is stored three times within an Azure storage cluster in the primary region. All writes are then asynchronously replicated to a Microsoft-defined secondary region. Geo-redundant storage provides 6 copies of your data spread between two Azure regions. In the event of a major disaster such as the permanent loss of an Azure region due to a natural disaster or other similar event, Microsoft will perform a failover so that the secondary in effect becomes the primary, serving all operations. Since the replication between the primary and secondary regions are asynchronous, in the event of a major disaster, data not yet replicated to the secondary region will be lost. You can also perform a manual failover of a geo-redundant storage account.
- **Geo-zone redundant:** Geo-zone redundant storage, often referred to as GZRS, is like zone redundant storage, in that a file is stored nine times across 3 distinct storage clusters in the primary region. All writes are then asynchronously replicated to a Microsoft-defined secondary region. The failover process for geo-zone-redundant storage works the same as it does for geo-redundant storage.

Standard Azure file shares support all four redundancy types, while premium Azure file shares only support locally redundant and zone redundant storage.

General purpose version 2 (GPv2) storage accounts provide two additional redundancy options that are not supported by Azure Files: read accessible geo-redundant storage, often referred to as RA-GRS, and read accessible geo-zone-redundant storage, often referred to as RA-GZRS. You can provision Azure file shares in storage accounts with these options set, however Azure Files does not support reading from the secondary region. Azure file shares deployed into read-accessible geo- or geo-zone redundant storage accounts will be billed as geo-redundant or geo-zone-redundant storage, respectively.

IMPORTANT

Geo-redundant and Geo-zone redundant storage have the capability to manually failover storage to the secondary region. We recommend that you do not do this outside of a disaster when you are using Azure File Sync because of the increased likelihood of data loss. In the event of a disaster where you would like to initiate a manual failover of storage, you will need to open up a support case with Microsoft to get Azure File Sync to resume sync with the secondary endpoint.

Migration

If you have an existing Windows file server, Azure File Sync can be directly installed in place, without the need to move data over to a new server. If you are planning to migrate to a new Windows file server as a part of adopting Azure File Sync, there are several possible approaches to move data over:

- Create server endpoints for your old file share and your new file share and let Azure File Sync synchronize the data between the server endpoints. The advantage to this approach is that it makes it very easy to oversubscribe the storage on your new file server, since Azure File Sync is cloud tiering aware. When you are ready, you can cut over end users to the file share on the new server and remove the old file share's server endpoint.
- Create a server endpoint only on the new file server, and copy data into from the old file share using `robocopy`. Depending on the topology of file shares on your new server (how many shares you have on each volume, how free each volume is, etc.), you may need to temporarily provision additional storage as it is expected that `robocopy` from your old server to your new server within your on-premises datacenter will complete faster than Azure File Sync will move data into Azure.

It is also possible to use Data Box to migrate data into an Azure File Sync deployment. Most of the time, when

customers want to use Data Box to ingest data, they do so because they think it will increase the speed of their deployment or because it will help with constrained bandwidth scenarios. While it's true that using a Data Box to ingest data into your Azure File Sync deployment will decrease bandwidth utilization, it will likely be faster for most scenarios to pursue an online data upload through one of the methods described above. To learn more about how to use Data Box to ingest data into your Azure File Sync deployment, see [Migrate data into Azure File Sync with Azure Data Box](#).

A common mistake customers make when migrating data into their new Azure File Sync deployment is to copy data directly into the Azure file share, rather than on their Windows file servers. Although Azure File Sync will identify all of the new files on the Azure file share, and sync them back to your Windows file shares, this is generally considerably slower than loading data through the Windows file server. Many Azure copy tools, such as AzCopy, have the additional downside of not copying all of the important metadata of a file such as timestamps and ACLs.

Antivirus

Because antivirus works by scanning files for known malicious code, an antivirus product might cause the recall of tiered files. In versions 4.0 and above of the Azure File Sync agent, tiered files have the secure Windows attribute FILE_ATTRIBUTE_RECALL_ON_DATA_ACCESS set. We recommend consulting with your software vendor to learn how to configure their solution to skip reading files with this attribute set (many do it automatically).

Microsoft's in-house antivirus solutions, Windows Defender and System Center Endpoint Protection (SCEP), both automatically skip reading files that have this attribute set. We have tested them and identified one minor issue: when you add a server to an existing sync group, files smaller than 800 bytes are recalled (downloaded) on the new server. These files will remain on the new server and will not be tiered since they do not meet the tiering size requirement (>64kb).

NOTE

Antivirus vendors can check compatibility between their product and Azure File Sync using the [Azure File Sync Antivirus Compatibility Test Suite](#), which is available for download on the Microsoft Download Center.

Backup

Like antivirus solutions, backup solutions might cause the recall of tiered files. We recommend using a cloud backup solution to back up the Azure file share instead of an on-premises backup product.

If you are using an on-premises backup solution, backups should be performed on a server in the sync group that has cloud tiering disabled. When performing a restore, use the volume-level or file-level restore options. Files restored using the file-level restore option will be synced to all endpoints in the sync group and existing files will be replaced with the version restored from backup. Volume-level restores will not replace newer file versions in the Azure file share or other server endpoints.

NOTE

Bare-metal (BMR) restore can cause unexpected results and is not currently supported.

NOTE

With Version 9 of the Azure File Sync agent, VSS snapshots (including Previous Versions tab) are now supported on volumes which have cloud tiering enabled. However, you must enable previous version compatibility through PowerShell. [Learn how.](#)

Azure File Sync agent update policy

The Azure File Sync agent is updated on a regular basis to add new functionality and to address issues. We recommend you configure Microsoft Update to get updates for the Azure File Sync agent as they're available.

Major vs. minor agent versions

- Major agent versions often contain new features and have an increasing number as the first part of the version number. For example: *2.*.**
- Minor agent versions are also called "patches" and are released more frequently than major versions. They often contain bug fixes and smaller improvements but no new features. For example: **.3.**

Upgrade paths

There are four approved and tested ways to install the Azure File Sync agent updates.

1. (Preferred) Configure Microsoft Update to automatically download and install agent updates.

We always recommend taking every Azure File Sync update to ensure you have access to the latest fixes for the server agent. Microsoft Update makes this process seamless, by automatically downloading and installing updates for you.

2. Use AfsUpdater.exe to download and install agent updates.

The AfsUpdater.exe is located in the agent installation directory. Double-click the executable to download and install agent updates.

3. Patch an existing Azure File Sync agent by using a Microsoft Update patch file, or a .msp executable. The latest Azure File Sync update package can be downloaded from the [Microsoft Update Catalog](#).

Running a .msp executable will upgrade your Azure File Sync installation with the same method used automatically by Microsoft Update in the previous upgrade path. Applying a Microsoft Update patch will perform an in-place upgrade of an Azure File Sync installation.

4. Download the newest Azure File Sync agent installer from the [Microsoft Download Center](#).

To upgrade an existing Azure File Sync agent installation, uninstall the older version and then install the latest version from the downloaded installer. The server registration, sync groups, and any other settings are maintained by the Azure File Sync installer.

Automatic agent lifecycle management

With agent version 6, the file sync team has introduced an agent auto-upgrade feature. You can select either of two modes and specify a maintenance window in which the upgrade shall be attempted on the server. This feature is designed to help you with the agent lifecycle management by either providing a guardrail preventing your agent from expiration or allowing for a no-hassle, stay current setting.

1. The **default setting** will attempt to prevent the agent from expiration. Within 21 days of the posted expiration date of an agent, the agent will attempt to self-upgrade. It will start an attempt to upgrade once a week within 21 days prior to expiration and in the selected maintenance window. **This option does not eliminate the need for taking regular Microsoft Update patches.**
2. Optionally, you can select that the agent will automatically upgrade itself as soon as a new agent version becomes available (currently not applicable to clustered servers). This update will occur during the selected maintenance window and allow your server to benefit from new features and improvements as soon as they become generally available. This is the recommended, worry-free setting that will provide major agent versions as well as regular update patches to your server. Every agent released is at GA quality. If you select

this option, Microsoft will flight the newest agent version to you. Clustered servers are excluded. Once flighting is complete, the agent will also become available on [Microsoft Download Center](#) aka.ms/AFS/agent.

Changing the auto-upgrade setting

The following instructions describe how to change the settings after you've completed the installer, if you need to make changes.

Open a PowerShell console and navigate to the directory where you installed the sync agent then import the server cmdlets. By default this would look something like this:

```
cd 'C:\Program Files\Azure\StorageSyncAgent'  
Import-Module -Name .\StorageSync.Management.ServerCmdlets.dll
```

You can run `Get-StorageSyncAgentAutoUpdatePolicy` to check the current policy setting and determine if you want to change it.

To change the current policy setting to the delayed update track, you can use:

```
Set-StorageSyncAgentAutoUpdatePolicy -PolicyMode UpdateBeforeExpiration
```

To change the current policy setting to the immediate update track, you can use:

```
Set-StorageSyncAgentAutoUpdatePolicy -PolicyMode InstallLatest
```

Agent lifecycle and change management guarantees

Azure File Sync is a cloud service, which continuously introduces new features and improvements. This means that a specific Azure File Sync agent version can only be supported for a limited time. To facilitate your deployment, the following rules guarantee you have enough time and notification to accommodate agent updates/upgrades in your change management process:

- Major agent versions are supported for at least six months from the date of initial release.
- We guarantee there is an overlap of at least three months between the support of major agent versions.
- Warnings are issued for registered servers using a soon-to-be expired agent at least three months prior to expiration. You can check if a registered server is using an older version of the agent under the registered servers section of a Storage Sync Service.
- The lifetime of a minor agent version is bound to the associated major version. For example, when agent version 3.0 is released, agent versions 2.* will all be set to expire together.

NOTE

Installing an agent version with an expiration warning will display a warning but succeed. Attempting to install or connect with an expired agent version is not supported and will be blocked.

Next steps

- [Consider firewall and proxy settings](#)
- [Planning for an Azure Files deployment](#)
- [Deploy Azure Files](#)
- [Deploy Azure File Sync](#)
- [Monitor Azure File Sync](#)

Azure Files networking considerations

2/25/2020 • 12 minutes to read • [Edit Online](#)

You can connect to an Azure file share in two ways:

- Accessing the share directly via the SMB or FileREST protocols. This access pattern is primarily employed when to eliminate as many on-premises servers as possible.
- Creating a cache of the Azure file share on an on-premises server with Azure File Sync, and accessing the file share's data from the on-premises server with your protocol of choice (SMB, NFS, FTPS, etc.) for your use case. This access pattern is handy because it combines the best of both on-premises performance and cloud scale and serverless attachable services, such as Azure Backup.

This article focuses on how to configure networking for when your use case calls for accessing the Azure file share directly rather than using Azure File Sync. For more information about networking considerations for an Azure File Sync deployment, see [configuring Azure File Sync proxy and firewall settings](#).

Networking configuration for Azure file shares is done on the Azure storage account. A storage account is a management construct that represents a shared pool of storage in which you can deploy multiple file shares, as well as other storage resources, such as blob containers or queues. Storage accounts expose multiple settings that help you secure network access to your file shares: network endpoints, storage account firewall settings, and encryption in transit.

Accessing your Azure file shares

When you deploy an Azure file share within a storage account, your file share is immediately accessible via the storage account's public endpoint. This means that authenticated requests, such as requests authorized by a user's logon identity, can originate securely from inside or outside of Azure.

In many customer environments, an initial mount of the Azure file share on your on-premises workstation will fail, even though mounts from Azure VMs succeed. The reason for this is that many organizations and internet service providers (ISPs) block the port that SMB uses to communicate, port 445. This practice originates from security guidance about legacy and deprecated versions of the SMB protocol. Although SMB 3.0 is an internet-safe protocol, older versions of SMB, especially SMB 1.0 are not. Azure file shares may only be externally accessed via SMB 3.0 and the FileREST protocol (which is also an internet safe protocol) via the public endpoint.

Since the easiest way to access your Azure file share from on-premises is to open your on-premises network to port 445, Microsoft recommends the following steps to remove SMB 1.0 from your environment:

1. Ensure that SMB 1.0 is removed or disabled on your organization's devices. All currently supported versions of Windows and Windows Server support removing or disabling SMB 1.0, and starting with Windows 10, version 1709, SMB 1.0 is not installed on the Windows by default. To learn more about how to disable SMB 1.0, see our OS-specific pages:
 - [Securing Windows/Windows Server](#)
 - [Securing Linux](#)
2. Ensure that no products within your organization require SMB 1.0 and remove the ones that do. We maintain an [SMB1 Product Clearinghouse](#), which contains all the first and third-party products known to Microsoft to require SMB 1.0.
3. (Optional) Use a third-party firewall with your organization's on-premises network to prevent SMB 1.0 traffic from leaving your organizational boundary.

If your organization requires port 445 to be blocked per policy or regulation, or your organization requires traffic

to Azure to follow a deterministic path, you can use Azure VPN Gateway or ExpressRoute to tunnel traffic to your Azure file shares.

IMPORTANT

Even if you decide use an alternate method to access your Azure file shares, Microsoft still recommends removing SMB 1.0 from your environment.

Tunneling traffic over a virtual private network or ExpressRoute

When you establish a network tunnel between your on-premises network and Azure, you are peering your on-premises network with one or more virtual networks in Azure. A [virtual network](#), or VNet, is similar to a traditional network that you'd operate on-premises. Like an Azure storage account or an Azure VM, a VNet is an Azure resource that is deployed in a resource group.

Azure Files supports the following mechanisms to tunnel traffic between your on-premises workstations and servers and Azure:

- [Azure VPN Gateway](#): A VPN gateway is a specific type of virtual network gateway that is used to send encrypted traffic between an Azure virtual network and an alternate location (such as on-premises) over the internet. An Azure VPN Gateway is an Azure resource that can be deployed in a resource group along side of a storage account or other Azure resources. VPN gateways expose two different types of connections:
 - [Point-to-Site \(P2S\) VPN](#) gateway connections, which are VPN connections between Azure and an individual client. This solution is primarily useful for devices that are not part of your organization's on-premises network, such as telecommuters who want to be able to mount their Azure file share from home, a coffee shop, or hotel while on the road. To use a P2S VPN connection with Azure Files, a P2S VPN connection will need to be configured for each client that wants to connect. To simplify the deployment of a P2S VPN connection, see [Configure a Point-to-Site \(P2S\) VPN on Windows for use with Azure Files](#) and [Configure a Point-to-Site \(P2S\) VPN on Linux for use with Azure Files](#).
 - [Site-to-Site \(S2S\) VPN](#), which are VPN connections between Azure and your organization's network. A S2S VPN connection enables you to configure a VPN connection once, for a VPN server or device hosted on your organization's network, rather than doing for every client device that needs to access your Azure file share. To simplify the deployment of a S2S VPN connection, see [Configure a Site-to-Site \(S2S\) VPN for use with Azure Files](#).
- [ExpressRoute](#), which enables you to create a defined route between Azure and your on-premises network that doesn't traverse the internet. Because ExpressRoute provides a dedicated path between your on-premises datacenter and Azure, ExpressRoute may be useful when network performance is a consideration. ExpressRoute is also a good option when your organization's policy or regulatory requirements require a deterministic path to your resources in the cloud.

Regardless of which tunneling method you use to access your Azure file shares, you need a mechanism to ensure the traffic to your storage account goes over the tunnel rather than your regular internet connection. It is technically possible to route to the public endpoint of the storage account, however this requires hard-coding all of the IP addresses for the Azure storage clusters in a region, since storage accounts may be moved between storage clusters at any time. This also requires constantly updating the IP address mappings since new clusters are added all the time.

Rather than hard-coding the IP address of your storage accounts into your VPN routing rules, we recommend using private endpoints, which give your storage account an IP address from the address space of an Azure virtual network. Since creating a tunnel to Azure establishes peering between your on-premises network and one or more virtual network, this enables the correct routing in a durable way.

Private endpoints

In addition to the default public endpoint for a storage account, Azure Files provides the option to have one or

more private endpoints. A private endpoint is an endpoint that is only accessible within an Azure virtual network. When you create a private endpoint for your storage account, your storage account gets a private IP address from within the address space of your virtual network, much like how an on-premises file server or NAS device receives an IP address within the dedicated address space of your on-premises network.

An individual private endpoint is associated with a specific Azure virtual network subnet. A storage account may have private endpoints in more than one virtual network.

Using private endpoints with Azure Files enables you to:

- Securely connect to your Azure file shares from on-premises networks using a VPN or ExpressRoute connection with private-peering.
- Secure your Azure file shares by configuring the storage account firewall to block all connections on the public endpoint. By default, creating a private endpoint does not block connections to the public endpoint.
- Increase security for the virtual network by enabling you to block exfiltration of data from the virtual network (and peering boundaries).

Private endpoints and DNS

When you create a private endpoint, by default we also create a (or update an existing) private DNS zone corresponding to the `privatelink` subdomain. Strictly speaking, creating a private DNS zone is not required to use a private endpoint for your storage account, but it is highly recommended in general and explicitly required when mounting your Azure file share with an Active Directory user principal or accessing from the FileREST API.

NOTE

This article uses the storage account DNS suffix for the Azure Public regions, `core.windows.net`. This commentary also applies to Azure Sovereign clouds such as the Azure US Government cloud and the Azure China cloud - just substitute the appropriate suffixes for your environment.

In your private DNS zone, we create an A record for `storageaccount.privatelink.file.core.windows.net` and a CNAME record for the regular name of the storage account, which follows the pattern `storageaccount.file.core.windows.net`. Since your Azure private DNS zone is connected to the virtual network containing the private endpoint, you can observe the DNS configuration when by calling the `Resolve-DnsName` cmdlet from PowerShell in an Azure VM (alternately `nslookup` in Windows and Linux):

```
Resolve-DnsName -Name "storageaccount.file.core.windows.net"
```

For this example, the storage account `storageaccount.file.core.windows.net` resolves to the private IP address of the private endpoint, which happens to be `192.168.0.4`.

Name	Type	TTL	Section	NameHost
storageaccount.file.core.windows.net	CNAME	29	Answer	csostoracct.privatelink.file.core.windows.net
Name : storageaccount.privatelink.file.core.windows.net				
QueryType : A				
TTL : 1769				
Section : Answer				
IP4Address : 192.168.0.4				
Name : privatelink.file.core.windows.net				
QueryType : SOA				
TTL : 269				
Section : Authority				
NameAdministrator : azureprivatedns-host.microsoft.com				
SerialNumber : 1				
TimeToZoneRefresh : 3600				
TimeToZoneFailureRetry : 300				
TimeToExpiration : 2419200				
DefaultTTL : 300				

If you run the same command from on-premises, you'll see that the same storage account name resolves to the public IP address of the storage account instead; `storageaccount.file.core.windows.net` is a CNAME record for `storageaccount.privatelink.file.core.windows.net`, which in turn is a CNAME record for the Azure storage cluster hosting the storage account:

Name	Type	TTL	Section	NameHost
storageaccount.file.core.windows.net	CNAME	60	Answer	storageaccount.privatelink.file.core.windows.net
storageaccount.privatelink.file.core.windows.net	CNAME	60	Answer	file.par20prdstr01a.store.core.windows.net
Name : file.par20prdstr01a.store.core.windows.net				
QueryType : A				
TTL : 60				
Section : Answer				
IP4Address : 52.239.194.40				

This reflects the fact that the storage account can expose both the public endpoint and one or more private endpoints. To ensure that the storage account name resolves to the private endpoint's private IP address, you must change the configuration on your on-premises DNS servers. This can be accomplished in several ways:

- Modifying the hosts file on your clients to make `storageaccount.file.core.windows.net` resolve to the desired private endpoint's private IP address. This is strongly discouraged for production environments, since you will need to make these changes to every client that wants to mount your Azure file shares and changes to the storage account or private endpoint will not be automatically handled.
- Creating an A record for `storageaccount.file.core.windows.net` in your on-premises DNS servers. This has the advantage that clients in your on-premises environment will be able to automatically resolve the storage account without needing to configure each client, however this solution is similarly brittle to modifying the hosts file because changes are not reflected. Although this solution is brittle, it may be the best choice for some environments.
- Forward the `.core.windows.net` zone from your on-premises DNS servers to your Azure private DNS zone. The Azure private DNS host can be reached through a special IP address (`168.63.129.16`) that is only accessible inside virtual networks that are linked to the Azure private DNS zone. To workaround this limitation, you can run additional DNS servers within your virtual network that will forward `.core.windows.net` on to the

Azure private DNS zone. To simplify this set up, we have provided PowerShell cmdlets that will auto-deploy DNS servers in your Azure virtual network and configure them as desired.

Storage account firewall settings

A firewall is a network policy which controls which requests are allowed to access the public endpoint for a storage account. Using the storage account firewall, you can restrict access to the storage account's public endpoint to certain IP addresses or ranges or to a virtual network. In general, most firewall policies for a storage account will restrict networking access to one or more virtual networks.

There are two approaches to restricting access to a storage account to a virtual network:

- Create one or more private endpoints for the storage account and restrict all access to the public endpoint. This ensures that only traffic originating from within the desired virtual networks can access the Azure file shares within the storage account.
- Restrict the public endpoint to one or more virtual networks. This works by using a capability of the virtual network called *service endpoints*. When you restrict the traffic to a storage account via a service endpoint, you are still accessing the storage account via the public IP address.

To learn more about how to configure the storage account firewall, see [configure Azure storage firewalls and virtual networks](#).

Encryption in transit

By default, all Azure storage accounts have encryption in transit enabled. This means that when you mount a file share over SMB or access it via the FileREST protocol (such as through the Azure portal, PowerShell/CLI, or Azure SDKs), Azure Files will only allow the connection if it is made with SMB 3.0+ with encryption or HTTPS. Clients that do not support SMB 3.0 or clients that support SMB 3.0 but not SMB encryption will not be able to mount the Azure file share if encryption in transit is enabled. For more information about which operating systems support SMB 3.0 with encryption, see our detailed documentation for [Windows](#), [macOS](#), and [Linux](#). All current versions of the PowerShell, CLI, and SDKs support HTTPS.

You can disable encryption in transit for an Azure storage account. When encryption is disabled, Azure Files will also allow SMB 2.1, SMB 3.0 without encryption, and un-encrypted FileREST API calls over HTTP. The primary reason to disable encryption in transit is to support a legacy application that must be run on an older operating system, such as Windows Server 2008 R2 or older Linux distribution. Azure Files only allows SMB 2.1 connections within the same Azure region as the Azure file share; an SMB 2.1 client outside of the Azure region of the Azure file share, such as on-premises or in a different Azure region, will not be able to access the file share.

For more information about encryption in transit, see [requiring secure transfer in Azure storage](#).

See also

- [Azure Files overview](#)
- [Planning for an Azure Files deployment](#)

Cloud Tiering Overview

1/28/2020 • 12 minutes to read • [Edit Online](#)

Cloud tiering is an optional feature of Azure File Sync in which frequently accessed files are cached locally on the server while all other files are tiered to Azure Files based on policy settings. When a file is tiered, the Azure File Sync file system filter (StorageSync.sys) replaces the file locally with a pointer, or reparse point. The reparse point represents a URL to the file in Azure Files. A tiered file has both the "offline" attribute and the FILE_ATTRIBUTE_RECALL_ON_DATA_ACCESS attribute set in NTFS so that third-party applications can securely identify tiered files.

When a user opens a tiered file, Azure File Sync seamlessly recalls the file data from Azure Files without the user needing to know that the file is stored in Azure.

IMPORTANT

Cloud tiering is not supported for server endpoints on the Windows system volumes, and only files greater than 64 KiB in size can be tiered to Azure Files.

Azure File Sync does not support tiering files smaller than 64 KiB as the performance overhead of tiering and recalling such small files would outweigh the space savings.

IMPORTANT

To recall files that have been tiered, the network bandwidth should be at least 1 Mbps. If network bandwidth is less than 1 Mbps, files may fail to recall with a timeout error.

Cloud Tiering FAQ

How does cloud tiering work?

The Azure File Sync system filter builds a "heatmap" of your namespace on each server endpoint. It monitors accesses (read and write operations) over time and then, based on both the frequency and recency of access, assigns a heat score to every file. A frequently accessed file that was recently opened will be considered hot, whereas a file that is barely touched and has not been accessed for some time will be considered cool. When the file volume on a server exceeds the volume free space threshold you set, it will tier the coolest files to Azure Files until your free space percentage is met.

In versions 4.0 and above of the Azure File Sync agent, you can additionally specify a date policy on each server endpoint that will tier any files not accessed or modified within a specified number of days.

How does the volume free space tiering policy work?

Volume free space is the amount of free space you wish to reserve on the volume on which a server endpoint is located. For example, if volume free space is set to 20% on a volume that has one server endpoint, up to 80% of the volume space will be occupied by the most recently accessed files, with any remaining files that do not fit into this space tiered up to Azure. Volume free space applies at the volume level rather than at the level of individual directories or sync groups.

How does the volume free space tiering policy work with regards to new server endpoints?

When a server endpoint is newly provisioned and connected to an Azure file share, the server will first pull down the namespace and then will pull down the actual files until it hits its volume free space threshold. This process is

also known as fast disaster recovery or rapid namespace restore.

How is volume free space interpreted when I have multiple server endpoints on a volume?

When there is more than one server endpoint on a volume, the effective volume free space threshold is the largest volume free space specified across any server endpoint on that volume. Files will be tiered according to their usage patterns regardless of which server endpoint to which they belong. For example, if you have two server endpoints on a volume, Endpoint1 and Endpoint2, where Endpoint1 has a volume free space threshold of 25% and Endpoint2 has a volume free space threshold of 50%, the volume free space threshold for both server endpoints will be 50%.

How does the date tiering policy work in conjunction with the volume free space tiering policy?

When enabling cloud tiering on a server endpoint, you set a volume free space policy. It always takes precedence over any other policies, including the date policy. Optionally, you can enable a date policy for each server endpoint on that volume, meaning that only files accessed (that is, read or written to) within the range of days this policy describes will be kept local, with any staler files tiered. Keep in mind that the volume free space policy always takes precedence, and when there isn't enough free space on the volume to retain as many days worth of files as described by the date policy, Azure File Sync will continue tiering the coldest files until the volume free space percentage is met.

For example, say you have a date-based tiering policy of 60 days and a volume free space policy of 20%. If, after applying the date policy, there is less than 20% of free space on the volume, the volume free space policy will kick in and override the date policy. This will result in more files being tiered, such that the amount of data kept on the server may be reduced from 60 days of data to 45 days. Conversely, this policy will force the tiering of files that fall outside of your time range even if you have not hit your free space threshold – so a file that is 61 days old will be tiered even if your volume is empty.

How do I determine the appropriate amount of volume free space?

The amount of data you should keep local is determined by a few factors: your bandwidth, your dataset's access pattern, and your budget. If you have a low-bandwidth connection, you may want to keep more of your data local to ensure there is minimal lag for your users. Otherwise, you can base it on the churn rate during a given period. For example, if you know that about 10% of your 1 TB dataset changes or is actively accessed each month, then you may want to keep 100 GB local so you are not frequently recalling files. If your volume is 2TB, then you will want to keep 5% (or 100 GB) local, meaning the remaining 95% is your volume free space percentage. However, we recommend that you add a buffer to account for periods of higher churn – in other words, starting with a lower volume free space percentage, and then adjusting it if needed later.

Keeping more data local means lower egress costs as fewer files will be recalled from Azure, but also requires you to maintain a larger amount of on-premises storage, which comes at its own cost. Once you have an instance of Azure File Sync deployed, you can look at your storage account's egress to roughly gauge whether your volume free space settings are appropriate for your usage. Assuming the storage account contains only your Azure File Sync Cloud Endpoint (that is, your sync share), then high egress means that many files are being recalled from the cloud, and you should consider increasing your local cache.

I've added a new server endpoint. How long until my files on this server tier?

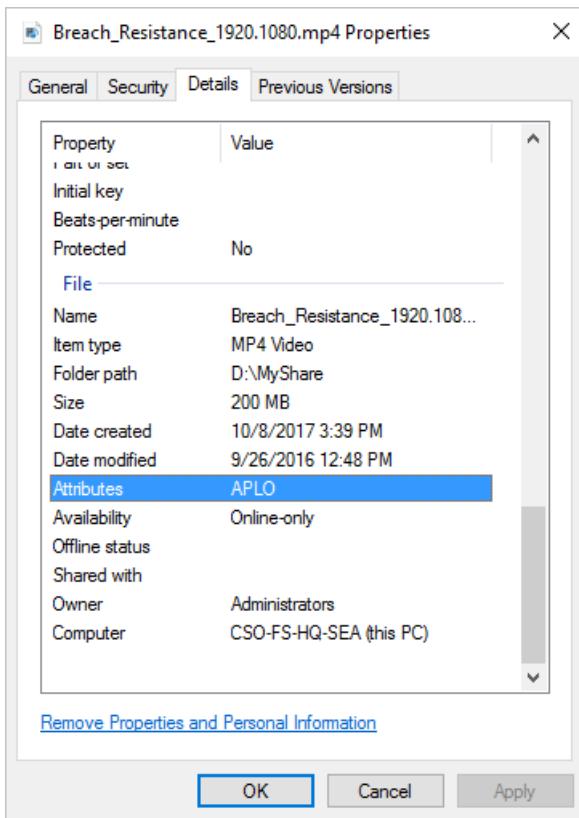
In versions 4.0 and above of the Azure File Sync agent, once your files have been uploaded to the Azure file share, they will be tiered according to your policies as soon as the next tiering session runs, which happen once an hour. On older agents, tiering can take up to 24 hours to happen.

How can I tell whether a file has been tiered?

There are several ways to check whether a file has been tiered to your Azure file share:

- **Check the file attributes on the file.** Right-click on a file, go to **Details**, and then scroll down to the **Attributes** property. A tiered file has the following attributes set:

ATTRIBUTE LETTER	ATTRIBUTE	DEFINITION
A	Archive	Indicates that the file should be backed up by backup software. This attribute is always set, regardless of whether the file is tiered or stored fully on disk.
P	Sparse file	Indicates that the file is a sparse file. A sparse file is a specialized type of file that NTFS offers for efficient use when the file on the disk stream is mostly empty. Azure File Sync uses sparse files because a file is either fully tiered or partially recalled. In a fully tiered file, the file stream is stored in the cloud. In a partially recalled file, that part of the file is already on disk. If a file is fully recalled to disk, Azure File Sync converts it from a sparse file to a regular file. This attribute is only set on Windows Server 2016 and older.
M	Recall on data access	Indicates that the file's data is not fully present on local storage. Reading the file will cause at least some of the file content to be fetched from an Azure file share to which the server endpoint is connected. This attribute is only set on Windows Server 2019.
L	Reparse point	Indicates that the file has a reparse point. A reparse point is a special pointer for use by a file system filter. Azure File Sync uses reparse points to define to the Azure File Sync file system filter (StorageSync.sys) the cloud location where the file is stored. This supports seamless access. Users won't need to know that Azure File Sync is being used or how to get access to the file in your Azure file share. When a file is fully recalled, Azure File Sync removes the reparse point from the file.
O	Offline	Indicates that some or all of the file's content is not stored on disk. When a file is fully recalled, Azure File Sync removes this attribute.



You can see the attributes for all the files in a folder by adding the **Attributes** field to the table display of File Explorer. To do this, right-click on an existing column (for example, **Size**), select **More**, and then select **Attributes** from the drop-down list.

- Use `fsutil` to check for reparse points on a file. As described in the preceding option, a tiered file always has a reparse point set. A reparse pointer is a special pointer for the Azure File Sync file system filter (StorageSync.sys). To check whether a file has a reparse point, in an elevated Command Prompt or PowerShell window, run the `fsutil` utility:

```
fsutil reparsepoint query <your-file-name>
```

If the file has a reparse point, you can expect to see **Reparse Tag Value: 0x8000001e**. This hexadecimal value is the reparse point value that is owned by Azure File Sync. The output also contains the reparse data that represents the path to your file on your Azure file share.

WARNING

The `fsutil reparsepoint` utility command also has the ability to delete a reparse point. Do not execute this command unless the Azure File Sync engineering team asks you to. Running this command might result in data loss.

A file I want to use has been tiered. How can I recall the file to disk to use it locally?

The easiest way to recall a file to disk is to open the file. The Azure File Sync file system filter (StorageSync.sys) seamlessly downloads the file from your Azure file share without any work on your part. For file types that can be partially read from, such as multimedia or .zip files, opening a file doesn't download the entire file.

You also can use PowerShell to force a file to be recalled. This option might be useful if you want to recall multiple files at once, such as all the files in a folder. Open a PowerShell session to the server node where Azure File Sync is installed, and then run the following PowerShell commands:

```
Import-Module "C:\Program Files\Azure\StorageSyncAgent\StorageSync.Management.ServerCmdlets.dll"
Invoke-StorageSyncFileRecall -Path <path-to-your-server-endpoint>
```

Optional parameters:

- `-Order CloudTieringPolicy` will recall the most recently modified files first.
- `-ThreadCount` determines how many files can be recalled in parallel.
- `-PerFileRetryCount` determines how often a recall will be attempted of a file that is currently blocked.
- `-PerFileRetryDelaySeconds` determines the time in seconds between retry to recall attempts and should always be used in combination with the previous parameter.

NOTE

If the local volume hosting the server does not have enough free space to recall all the tiered data, the

```
Invoke-StorageSyncFileRecall
```

Why doesn't the **Size on disk** property for a file match the **Size** property after using Azure File Sync?

Windows File Explorer exposes two properties to represent the size of a file: **Size** and **Size on disk**. These properties differ subtly in meaning. **Size** represents the complete size of the file. **Size on disk** represents the size of the file stream that's stored on the disk. The values for these properties can differ for a variety of reasons, such as compression, use of Data Deduplication, or cloud tiering with Azure File Sync. If a file is tiered to an Azure file share, the size on the disk is zero, because the file stream is stored in your Azure file share, and not on the disk. It's also possible for a file to be partially tiered (or partially recalled). In a partially tiered file, part of the file is on disk. This might occur when files are partially read by applications like multimedia players or zip utilities.

How do I force a file or directory to be tiered?

When the cloud tiering feature is enabled, cloud tiering automatically tiers files based on last access and modify times to achieve the volume free space percentage specified on the cloud endpoint. Sometimes, though, you might want to manually force a file to tier. This might be useful if you save a large file that you don't intend to use again for a long time, and you want the free space on your volume now to use for other files and folders. You can force tiering by using the following PowerShell commands:

```
Import-Module "C:\Program Files\Azure\StorageSyncAgent\StorageSync.Management.ServerCmdlets.dll"
Invoke-StorageSyncCloudTiering -Path <file-or-directory-to-be-tiered>
```

Why are my tiered files not showing thumbnails or previews in Windows Explorer?

For tiered files, thumbnails and previews won't be visible at your server endpoint. This behavior is expected since the thumbnail cache feature in Windows intentionally skips reading files with the offline attribute. With Cloud Tiering enabled, reading through tiered files would cause them to be downloaded (recalled).

This behavior is not specific to Azure File Sync, Windows Explorer displays a "grey X" for any files that have the offline attribute set. You will see the X icon when accessing files over SMB. For a detailed explanation of this behavior, refer to <https://blogs.msdn.microsoft.com/oldnewthing/20170503-00/?p=96105>

Next Steps

- [Planning for an Azure File Sync Deployment](#)

2 minutes to read

Overview of share snapshots for Azure Files

10/15/2019 • 5 minutes to read • [Edit Online](#)

Azure Files provides the capability to take share snapshots of file shares. Share snapshots capture the share state at that point in time. In this article, we describe what capabilities share snapshots provide and how you can take advantage of them in your custom use case.

When to use share snapshots

Protection against application error and data corruption

Applications that use file shares perform operations such as writing, reading, storage, transmission, and processing. If an application is misconfigured or an unintentional bug is introduced, accidental overwrite or damage can happen to a few blocks. To help protect against these scenarios, you can take a share snapshot before you deploy new application code. If a bug or application error is introduced with the new deployment, you can go back to a previous version of your data on that file share.

Protection against accidental deletions or unintended changes

Imagine that you're working on a text file in a file share. After the text file is closed, you lose the ability to undo your changes. In these cases, you then need to recover a previous version of the file. You can use share snapshots to recover previous versions of the file if it's accidentally renamed or deleted.

General backup purposes

After you create a file share, you can periodically create a share snapshot of the file share to use it for data backup. A share snapshot, when taken periodically, helps maintain previous versions of data that can be used for future audit requirements or disaster recovery.

Capabilities

A share snapshot is a point-in-time, read-only copy of your data. You can create, delete, and manage snapshots by using the REST API. Same capabilities are also available in the client library, Azure CLI, and Azure portal.

You can view snapshots of a share by using both the REST API and SMB. You can retrieve the list of versions of the directory or file, and you can mount a specific version directly as a drive (only available on Windows - see [Limits](#)).

After a share snapshot is created, it can be read, copied, or deleted, but not modified. You can't copy a whole share snapshot to another storage account. You have to do that file by file, by using AzCopy or other copying mechanisms.

Share snapshot capability is provided at the file share level. Retrieval is provided at individual file level, to allow for restoring individual files. You can restore a complete file share by using SMB, the REST API, the portal, the client library, or PowerShell/CLI tooling.

A share snapshot of a file share is identical to its base file share. The only difference is that a **DateTime** value is appended to the share URI to indicate the time at which the share snapshot was taken. For example, if a file share URI is <http://storagesample.core.file.windows.net/myshare>, the share snapshot URI is similar to:

```
http://storagesample.core.file.windows.net/myshare?snapshot=2011-03-09T01:42:34.9360000Z
```

Share snapshots persist until they are explicitly deleted. A share snapshot cannot outlive its base file share. You can enumerate the snapshots associated with the base file share to track your current snapshots.

When you create a share snapshot of a file share, the files in the share's system properties are copied to the share snapshot with the same values. The base files and the file share's metadata are also copied to the share snapshot, unless you specify separate metadata for the share snapshot when you create it.

You cannot delete a share that has share snapshots unless you delete all the share snapshots first.

Space usage

Share snapshots are incremental in nature. Only the data that has changed after your most recent share snapshot is saved. This minimizes the time required to create the share snapshot and saves on storage costs. Any write operation to the object or property or metadata update operation is counted toward "changed content" and is stored in the share snapshot.

To conserve space, you can delete the share snapshot for the period when the churn was highest.

Even though share snapshots are saved incrementally, you need to retain only the most recent share snapshot in order to restore the share. When you delete a share snapshot, only the data unique to that share snapshot is removed. Active snapshots contain all the information that you need to browse and restore your data (from the time the share snapshot was taken) to the original location or an alternate location. You can restore at the item level.

Snapshots don't count toward your 5-TB share limit. There is no limit to how much space share snapshots occupy in total. Storage account limits still apply.

Limits

The maximum number of share snapshots that Azure Files allows today is 200. After 200 share snapshots, you have to delete older share snapshots in order to create new ones.

There is no limit to the simultaneous calls for creating share snapshots. There is no limit to amount of space that share snapshots of a particular file share can consume.

Today, it is not possible to mount share snapshots on Linux. This is because the Linux SMB client does not support mounting snapshots like Windows does.

Copying data back to a share from share snapshot

Copy operations that involve files and share snapshots follow these rules:

You can copy individual files in a file share snapshot over to its base share or any other location. You can restore an earlier version of a file or restore the complete file share by copying file by file from the share snapshot. The share snapshot is not promoted to base share.

The share snapshot remains intact after copying, but the base file share is overwritten with a copy of the data that was available in the share snapshot. All the restored files count toward "changed content."

You can copy a file in a share snapshot to a different destination with a different name. The resulting destination file is a writable file and not a share snapshot. In this case, your base file share will remain intact.

When a destination file is overwritten with a copy, any share snapshots associated with the original destination file remain intact.

General best practices

When you're running infrastructure on Azure, automate backups for data recovery whenever possible. Automated actions are more reliable than manual processes, helping to improve data protection and recoverability. You can use the REST API, the Client SDK, or scripting for automation.

Before you deploy the share snapshot scheduler, carefully consider your share snapshot frequency and retention settings to avoid incurring unnecessary charges.

Share snapshots provide only file-level protection. Share snapshots don't prevent fat-finger deletions on a file share or storage account. To help protect a storage account from accidental deletions, you can lock the storage account or the resource group.

Next steps

- Working with share snapshots in:

- [PowerShell](#)
- [CLI](#)
- [Windows](#)
- [Share snapshot FAQ](#)

Azure Files scalability and performance targets

2/25/2020 • 10 minutes to read • [Edit Online](#)

[Azure Files](#) offers fully managed file shares in the cloud that are accessible via the industry standard SMB protocol. This article discusses the scalability and performance targets for Azure Files and Azure File Sync.

The scalability and performance targets listed here are high-end targets, but may be affected by other variables in your deployment. For example, the throughput for a file may also be limited by your available network bandwidth, not just the servers hosting the Azure Files service. We strongly recommend testing your usage pattern to determine whether the scalability and performance of Azure Files meet your requirements. We are also committed to increasing these limits over time. Please don't hesitate to give us feedback, either in the comments below or on the [Azure Files UserVoice](#), about which limits you would like to see us increase.

Azure storage account scale targets

The parent resource for an Azure file share is an Azure storage account. A storage account represents a pool of storage in Azure that can be used by multiple storage services, including Azure Files, to store data. Other services that store data in storage accounts are Azure Blob storage, Azure Queue storage, and Azure Table storage. The following targets apply all storage services storing data in a storage account:

The following table describes default limits for Azure general-purpose v1, v2, and Blob storage accounts. The *ingress* limit refers to all data from requests that are sent to a storage account. The *egress* limit refers to all data from responses that are received from a storage account.

RESOURCE	DEFAULT LIMIT
Number of storage accounts per region per subscription, including both standard and premium accounts	250
Maximum storage account capacity	2 PiB for US and Europe, and 500 TiB for all other regions (including the UK) ¹
Maximum number of blob containers, blobs, file shares, tables, queues, entities, or messages per storage account	No limit
Maximum request rate ¹ per storage account	20,000 requests per second
Maximum ingress ¹ per storage account (US, Europe regions)	25 Gbps
Maximum ingress ¹ per storage account (regions other than US and Europe)	5 Gbps if RA-GRS/GRS is enabled, 10 Gbps for LRS/ZRS ²
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 ²
Maximum egress for general-purpose v1 storage accounts (non-US regions)	10 Gbps if RA-GRS/GRS is enabled, 15 Gbps for LRS/ZRS ²

RESOURCE	DEFAULT LIMIT
Maximum number of virtual network rules per storage account	200
Maximum number of IP address rules per storage account	200

¹Azure Storage standard accounts support higher capacity limits and higher limits for ingress by request. To request an increase in account limits for ingress, contact [Azure Support](#). For more information, see [Announcing larger, higher scale storage accounts](#).

²If 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 \(RA-GZRS\)](#)

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 and support the scalability and performance targets outlined in this article, 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](#).

The following limits apply only when you perform management operations by using Azure Resource Manager with Azure Storage.

RESOURCE	DEFAULT LIMIT
Storage account management operations (read)	800 per 5 minutes
Storage account management operations (write)	1200 per hour
Storage account management operations (list)	100 per 5 minutes

IMPORTANT

General purpose storage account utilization from other storage services affects your Azure file shares in your storage account. For example, if you reach the maximum storage account capacity with Azure Blob storage, you will not be able to create new files on your Azure file share, even if your Azure file share is below the maximum share size.

Azure Files scale targets

There are three categories of limitations to consider for Azure Files: storage accounts, shares, and files.

For example: With premium file shares, a single share can achieve 100,000 IOPS and a single file can scale up to 5,000 IOPS. So, if you have three files in one share, the maximum IOPS you can get from that share is 15,000.

Standard storage account limits

See the [Azure storage account scale targets](#) section for these limits.

Premium FileStorage account limits

Premium files use a unique storage account called **FileStorage**. This account type is designed for workloads with high IOPS, high throughput with consistent low-latency. Premium file storage scales with the provisioned share size.

AREA	TARGET
Max provisioned size	100 TiB
Shares	Unlimited
IOPS	100,000
Ingress	4,136 MiB/s
Egress	6,204 MiB/s

IMPORTANT

Storage account limits apply to all shares. Scaling up to the max for FileStorage accounts is only achievable if there is only one share per FileStorage account.

File share and file scale targets

NOTE

Standard file shares larger than 5 TiB have certain limitations and regional restrictions. For a list of limitations, regional information, and instructions to enable these larger file share sizes, see the [Onboard to larger file shares](#) section of the planning guide.

RESOURCE	STANDARD FILE SHARES	PREMIUM FILE SHARES
Minimum size of a file share	No minimum; pay as you go	100 GiB; provisioned
Maximum size of a file share	100 TiB*, 5 TiB	100 TiB
Maximum size of a file in a file share	1 TiB	1 TiB
Maximum number of files in a file share	No limit	No limit
Maximum IOPS per share	10,000 IOPS*, 1,000 IOPS	100,000 IOPS

RESOURCE	STANDARD FILE SHARES	PREMIUM FILE SHARES
Maximum number of stored access policies per file share	5	5
Target throughput for a single file share	up to 300 MiB/sec*, Up to 60 MiB/sec ,	See premium file share ingress and egress values
Maximum egress for a single file share	See standard file share target throughput	Up to 6,204 MiB/s
Maximum ingress for a single file share	See standard file share target throughput	Up to 4,136 MiB/s
Maximum open handles per file	2,000 open handles	2,000 open handles
Maximum number of share snapshots	200 share snapshots	200 share snapshots
Maximum object (directories and files) name length	2,048 characters	2,048 characters
Maximum pathname component (in the path \A\B\C\D, each letter is a component)	255 characters	255 characters

* Available in most regions, see [Regional availability](#) for the details on available regions.

Additional premium file share level limits

AREA	TARGET
Minimum size increase/decrease	1 GiB
Baseline IOPS	1 IOPS per GiB, up to 100,000
IOPS bursting	3x IOPS per GiB, up to 100,000
Egress rate	60 MiB/s + 0.06 * provisioned GiB
Ingress rate	40 MiB/s + 0.04 * provisioned GiB

File level limits

AREA	PREMIUM FILE	STANDARD FILE
Size	1 TiB	1 TiB
Max IOPS per file	5,000	1,000
Concurrent handles	2,000	2,000
Egress	300 MiB/sec	See standard file throughput values
Ingress	200 MiB/sec	See standard file throughput values

Area	Premium File	Standard File
Throughput	See premium file ingress/egress values	Up to 60 MiB/sec

Azure File Sync scale targets

Azure File Sync has been designed with the goal of limitless usage, but limitless usage is not always possible. The following table indicates the boundaries of Microsoft's testing and also indicates which targets are hard limits:

Resource	Target	Hard Limit
Storage Sync Services per region	20 Storage Sync Services	Yes
Sync groups per Storage Sync Service	100 sync groups	Yes
Registered servers per Storage Sync Service	99 servers	Yes
Cloud endpoints per sync group	1 cloud endpoint	Yes
Server endpoints per sync group	50 server endpoints	No
Server endpoints per server	30 server endpoints	Yes
File system objects (directories and files) per sync group	100 million objects	No
Maximum number of file system objects (directories and files) in a directory	5 million objects	Yes
Maximum object (directories and files) security descriptor size	64 KiB	Yes
File size	100 GiB	No
Minimum file size for a file to be tiered	V9: Based on file system cluster size (double file system cluster size). For example, if the file system cluster size is 4kb, the minimum file size will be 8kb. V8 and older: 64 KiB	Yes

NOTE

An Azure File Sync endpoint can scale up to the size of an Azure file share. If the Azure file share size limit is reached, sync will not be able to operate.

Azure File Sync performance metrics

Since the Azure File Sync agent runs on a Windows Server machine that connects to the Azure file shares, the effective sync performance depends upon a number of factors in your infrastructure: Windows Server and the underlying disk configuration, network bandwidth between the server and the Azure storage, file size, total dataset size, and the activity on the dataset. Since Azure File Sync works on the file level, the performance characteristics of an Azure File Sync-based solution is better measured in the number of objects (files and directories) processed per second.

For Azure File Sync, performance is critical in two stages:

1. **Initial one-time provisioning:** To optimize performance on initial provisioning, refer to [Onboarding with Azure File Sync](#) for the optimal deployment details.
2. **Ongoing sync:** After the data is initially seeded in the Azure file shares, Azure File Sync keeps multiple endpoints in sync.

To help you plan your deployment for each of the stages, below are the results observed during the internal testing on a system with a config

SYSTEM CONFIGURATION	
CPU	64 Virtual Cores with 64 MiB L3 cache
Memory	128 GiB
Disk	SAS disks with RAID 10 with battery backed cache
Network	1 Gbps Network
Workload	General Purpose File Server

INITIAL ONE-TIME PROVISIONING	
Number of objects	25 million objects
Dataset Size	~4.7 TiB
Average File Size	~200 KiB (Largest File: 100 GiB)
Upload Throughput	20 objects per second per sync group
Namespace Download Throughput*	400 objects per second

*When a new server endpoint is created, the Azure File Sync agent does not download any of the file content. It first syncs the full namespace and then triggers background recall to download the files, either in their entirety or, if cloud tiering is enabled, to the cloud tiering policy set on the server endpoint.

ONGOING SYNC	
Number of objects synced	125,000 objects (~1% churn)
Dataset Size	50 GiB
Average File Size	~500 KiB
Upload Throughput	20 objects per second per sync group
Full Download Throughput*	60 objects per second

*If cloud tiering is enabled, you are likely to observe better performance as only some of the file data is downloaded. Azure File Sync only downloads the data of cached files when they are changed on any of the endpoints. For any tiered or newly created files, the agent does not download the file data, and instead only syncs

the namespace to all the server endpoints. The agent also supports partial downloads of tiered files as they are accessed by the user.

NOTE

The numbers above are not an indication of the performance that you will experience. The actual performance will depend on multiple factors as outlined in the beginning of this section.

As a general guide for your deployment, you should keep a few things in mind:

- The object throughput approximately scales in proportion to the number of sync groups on the server. Splitting data into multiple sync groups on a server yields better throughput, which is also limited by the server and network.
- The object throughput is inversely proportional to the MiB per second throughput. For smaller files, you will experience higher throughput in terms of the number of objects processed per second, but lower MiB per second throughput. Conversely, for larger files, you will get fewer objects processed per second, but higher MiB per second throughput. The MiB per second throughput is limited by the Azure Files scale targets.

See also

- [Planning for an Azure Files deployment](#)
- [Planning for an Azure File Sync deployment](#)

Overview of Azure Files identity-based authentication support for SMB access

2/25/2020 • 9 minutes to read • [Edit Online](#)

Azure Files supports identity-based authentication over Server Message Block (SMB) through [Active Directory \(AD\)](#) (preview) and [Azure Active Directory Domain Services \(Azure AD DS\)](#) (GA). This article focuses on how Azure Files can leverage domain services, either on-premises or in Azure, to support identity-based access to Azure Files over SMB. This allows you to easily replace your existing file servers with Azure Files and continue to use your existing directory service, maintaining seamless user access to shares.

Azure Files enforces authorization on the user access to both the share and the directory/file level. Share-level permission assignment can be assigned to Azure AD users or groups managed through the typical [role-based access control \(RBAC\)](#) model. With RBAC, the credentials you use for file access should be available or synced to Azure AD. You can assign built-in RBAC roles like Storage File Data SMB Share Reader to users or groups in Azure AD to grant read access to an Azure file share.

At the directory/file level, Azure Files supports preserving, inheriting, and enforcing [Windows DACLs](#) just like any Windows file servers. If you copy data over SMB from a file share to Azure Files, or vice versa, you can choose to keep Windows DACLs. Whether you plan to enforce authorization or not, you can leverage Azure Files to backup ACLs along with your data.

To learn how to enable AD authentication for Azure file shares, see [Enable Active Directory authentication over SMB for Azure file shares](#).

To learn how to enable Azure AD DS authentication for Azure file shares, see [Enable Azure Active Directory Domain Services authentication on Azure Files](#).

Glossary

It's helpful to understand some key terms relating to Azure AD Domain Service authentication over SMB for Azure file shares:

- **Kerberos authentication**

Kerberos is an authentication protocol that is used to verify the identity of a user or host. For more information on Kerberos, see [Kerberos Authentication Overview](#).

- **Server Message Block (SMB) protocol**

SMB is an industry-standard network file-sharing protocol. SMB is also known as Common Internet File System or CIFS. For more information on SMB, see [Microsoft SMB Protocol and CIFS Protocol Overview](#).

- **Azure Active Directory (Azure AD)**

Azure Active Directory (Azure AD) is Microsoft's multi-tenant cloud-based directory and identity management service. Azure AD combines core directory services, application access management, and identity protection into a single solution. Azure AD enables your domain-joined Windows virtual machines (VMs) to access Azure file shares with your Azure AD credentials. For more information, see [What is Azure Active Directory?](#)

- **Azure AD Domain Services (Azure AD DS)**

Azure AD Domain Services (GA) provides managed domain services such as domain join, group policies,

LDAP, and Kerberos/NTLM authentication. These services are fully compatible with Windows Server Active Directory. For more information, see [Azure Active Directory \(AD\) Domain Services](#).

- **Active Directory Domain Services (AD DS, also referred as AD)**

Active directory (AD) (preview) provides the methods for storing directory data while making it available to network users and administrators. Security is integrated with Active Directory through logon authentication and access control to objects in the directory. With a single network logon, administrators can manage directory data and organization throughout their network, and authorized network users can access resources anywhere on the network. AD is commonly adopted by enterprises in on-premises and use AD credentials as the identity for access control. For more information, see [Active Directory Domain Services Overview](#).

- **Azure Role Based Access Control (RBAC)**

Azure Role-Based Access Control (RBAC) enables fine-grained access management for Azure. Using RBAC, you can manage access to resources by granting users the fewest permissions needed to perform their jobs. For more information on RBAC, see [What is role-based access control \(RBAC\) in Azure?](#).

Common use cases

Identity-based authentication and support for Windows ACLs on Azure Files is best leveraged for the following use cases:

Replace on-premises file servers

Deprecating and replacing scattered on-premises file servers is a common problem that every enterprise encounters in their IT modernization journey. Azure file shares with AD (preview) authentication is the best fit here, when you can migrate the data to Azure Files. A complete migration will allow you to take advantage of the high availability and scalability benefits while also minimizing the client side changes, especially complicated AD domain infrastructure. It provides a seamless migration experience to end users, so they can continue to access their data with the same credentials using their existing domain joined machines.

Lift and shift applications to Azure

When you "lift and shift" applications to the cloud, you want to keep the same authentication model for your data. As we extend the identity-based access control experience to Azure file shares, it eliminates the need to change your application to modern auth methods and expedite cloud adoption. Azure file shares provides the option to integrate with either Azure AD DS (GA) or AD (preview) for authentication. If your plan is to be 100% cloud native and minimize the efforts managing cloud infrastructures, Azure AD DS would be a better fit as a fully managed domain service. If you need full compatibility with AD DS (GA) capabilities, you may want to consider extending your AD environment to cloud by self-hosting domain controllers on VMs. Either way, we provide the flexibility to choose the domain services that suits your business needs.

Backup and disaster recovery (DR)

If you are keeping your primary file storage on-premises, Azure file shares can serve as an ideal storage for backup or DR, to improve business continuity. You can use Azure file shares to back up your data from existing file servers, while preserving Windows DACLs. For DR scenarios, you can configure an authentication option to support proper access control enforcement at failover.

Supported scenarios

The following table summarizes the supported Azure file shares authentication scenarios for Azure AD DS (GA) and AD (preview). We recommend selecting the domain service that you adopted for your client environment for integration with Azure Files. If you have AD (preview) already setup on-premises or on Azure where your devices are domain joined to AD, you should choose to leverage AD (preview) for Azure file shares authentication. Similarly, if you've already adopted Azure AD DS (GA), you should use that for Azure file shares authentication.

AZURE AD DS (GA) AUTHENTICATION	AD (PREVIEW) AUTHENTICATION
Azure AD DS domain joined Windows machines can access Azure file shares with Azure AD credentials over SMB.	AD domain joined Windows machines can access Azure file shares with AD credentials that are synched to Azure AD over SMB.

Unsupported scenarios

- Azure AD DS (GA) and AD (preview) authentication do not support authentication against computer accounts. You can consider using a service logon account instead.
- Azure AD DS (GA) authentication does not support authentication against Azure AD cloud joined devices.

Advantages of identity-based authentication

Identity-based authentication for Azure Files offers several benefits over using Shared Key authentication:

- **Extend the traditional identity-based file share access experience to the cloud with AD and Azure AD DS**

If you plan to "lift and shift" your application to the cloud, replacing traditional file servers with Azure file shares, then you may want your application to authenticate with either AD or Azure AD credentials to access file data. Azure Files supports using both AD or Azure AD credentials to access Azure file shares over SMB from either AD or Azure AD DS domain-joined VMs.

- **Enforce granular access control on Azure file shares**

You can grant permissions to a specific identity at the share, directory, or file level. For example, suppose that you have several teams using a single Azure file share for project collaboration. You can grant all teams access to non-sensitive directories, while limiting access to directories containing sensitive financial data to your Finance team only.

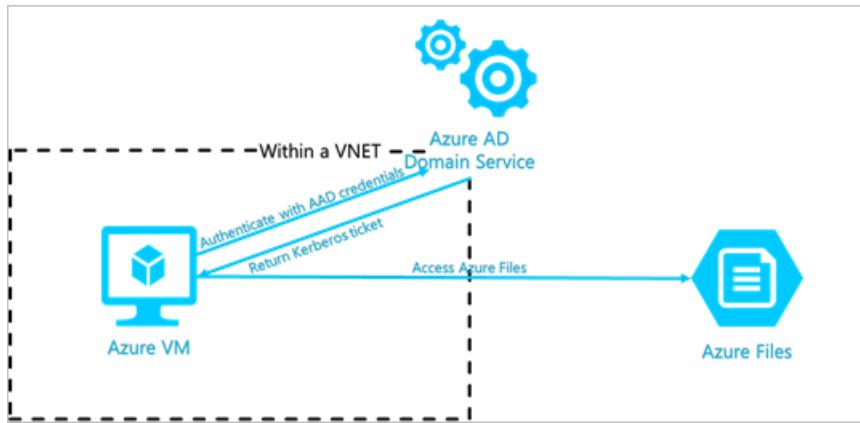
- **Back up Windows ACLs (also known as NTFS) along with your data**

You can use Azure file shares to back up your existing on-premises file shares. Azure Files preserves your ACLs along with your data when you back up a file share to Azure file shares over SMB.

How it works

Azure file shares supports kerberos authentication for integration with either Azure AD DS (GA) or AD (preview). Before you can enable authentication on Azure file shares, you must first setup your domain environment. For Azure AD DS (GA) authentication, you should enable Azure AD Domain Services and domain join the VMs you plan to access file data from. Your domain-joined VM must reside in the same virtual network (VNET) as your Azure AD Domain Services. Similarly, for AD (preview) authentication, you need to setup your Active Directory domain controller and domain join your machines or VMs.

When an identity associated with an application running on a VM attempts to access data in Azure file shares, the request is sent to Azure AD Domain Services to authenticate the identity. If authentication is successful, Azure AD Domain Services returns a Kerberos token. The application sends a request that includes the Kerberos token, and Azure file shares use that token to authorize the request. Azure file shares receives the token only and does not persist Azure AD credentials. AD authentication works in a similar fashion, where AD provides the Kerberos token.



Enable identity-based authentication

You can enable identity-based authentication with either Azure AD DS (GA) or AD (preview) for Azure file shares on your new and existing storage accounts. Only one domain service can be used for file access authentication on the storage account, which applies to all file shares in the account. Detailed step by step guidance on setting up you file shares for authentication with Azure AD DS (GA) in our article [Enable Azure Active Directory Domain Services authentication on Azure Files](#) and guidance for AD (preview) in our other article, [Enable Active Directory authentication over SMB for Azure file shares](#).

Configure share-level permissions for Azure Files

Once either Azure AD DS (GA) or AD (preview) authentication is enabled, you can use built-in RBAC roles or configure custom roles for Azure AD identities and assign access rights to any file shares in your storage accounts. the assigned permission allows the granted identity to get access to the share only, nothing else, not even the root directory. You still need to separately configure directory or file-level permissions for Azure file shares.

Configure directory or file-level permissions for Azure Files

Azure file shares enforces standard Windows file permissions at both the directory and file level, including the root directory. Configuration of directory or file-level permissions is supported over both SMB and REST. Mount the target file share from your VM and configure permissions using Windows File Explorer, Windows `icacls`, or the `Set-ACL` command.

Use the storage account key for superuser permissions

A user possessing the storage account key can access Azure file shares with superuser permissions. Superuser permissions bypass all access control restrictions.

IMPORTANT

Our recommended security best practice is to avoid sharing your storage account keys, and leverage identity-based authentication whenever possible.

Preserve directory and file ACLs when importing data to Azure file shares

Azure Files supports preserving directory or file level ACLs when copying data to Azure file shares. You can copy ACLs on a directory or file to Azure file shares using either Azure File Sync or common file movement toolsets. For example, you can use `robocopy` with the `/copy:s` flag to copy data as well as ACLs to an Azure file share. ACLs are preserved by default, you are not required to enable identity-based authentication on your storage account to preserve ACLs.

Pricing

There is no additional service charge to enable identity-based authentication over SMB on your storage account. For more information on pricing, see [Azure Files pricing](#) and [Azure AD Domain Services pricing](#) pages if you are looking for AAD DS information.

Next steps

For more information about Azure Files and identity-based authentication over SMB, see these resources:

- [Planning for an Azure Files deployment](#)
- [Enable Active Directory authentication over SMB for Azure file shares](#)
- [Enable Azure Active Directory Domain Services authentication on Azure Files](#)
- [FAQ](#)

Azure Storage encryption for data at rest

2/7/2020 • 10 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 helps 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 new 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.

Storage accounts are encrypted regardless of their performance tier (standard or premium) or deployment model (Azure Resource Manager or classic). All Azure Storage redundancy options support encryption, and all copies of a storage account are encrypted. All Azure Storage resources are encrypted, including blobs, disks, files, queues, and tables. All object metadata is also encrypted.

Encryption does not affect Azure Storage performance. There is no additional cost for Azure Storage encryption.

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

You can rely on Microsoft-managed keys for the encryption of your storage account, or you can manage encryption with your own keys. If you choose to manage encryption with your own keys, you have two options:

- You can specify a *customer-managed key* with Azure Key Vault to use for encrypting and decrypting data in Blob storage and in Azure Files.^{1,2}
- You can specify a *customer-provided key* on Blob storage operations. A client making a read or write request against Blob storage can include an encryption key on the request for granular control over how blob data is encrypted and decrypted.

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

	MICROSOFT-MANAGED KEYS	CUSTOMER-MANAGED KEYS	CUSTOMER-PROVIDED KEYS
Encryption/decryption operations	Azure	Azure	Azure
Azure Storage services supported	All	Blob storage, Azure Files ^{1,2}	Blob storage

	MICROSOFT-MANAGED KEYS	CUSTOMER-MANAGED KEYS	CUSTOMER-PROVIDED KEYS
Key storage	Microsoft key store	Azure Key Vault	Azure Key Vault or any other key store
Key rotation responsibility	Microsoft	Customer	Customer
Key usage	Microsoft	Azure portal, Storage Resource Provider REST API, Azure Storage management libraries, PowerShell, CLI	Azure Storage REST API (Blob storage), Azure Storage client libraries
Key access	Microsoft only	Microsoft, Customer	Customer only

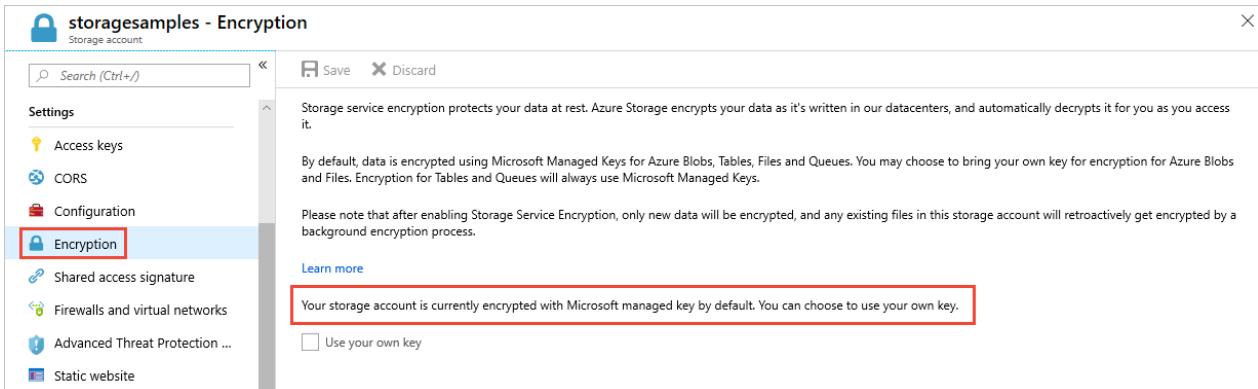
¹ For information about creating an account that supports using customer-managed keys with Queue storage, see [Create an account that supports customer-managed keys for queues](#).

² 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](#).

The following sections describe each of the options for key management in greater detail.

Microsoft-managed keys

By default, your storage account uses Microsoft-managed encryption keys. You can see the encryption settings for your storage account in the **Encryption** section of the [Azure portal](#), as shown in the following image.

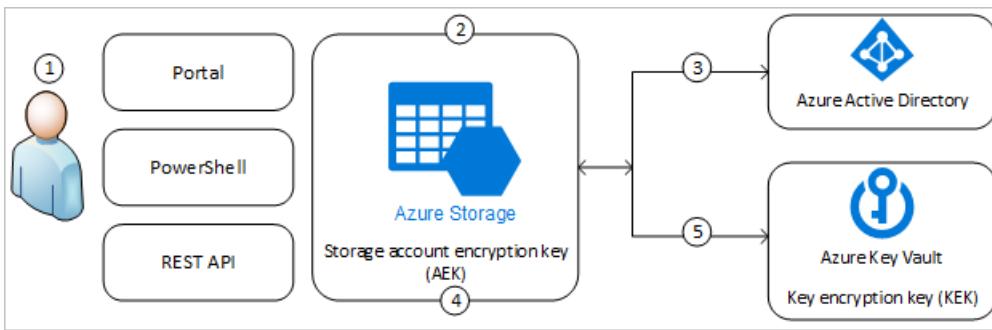


Customer-managed keys with Azure Key Vault

You can manage Azure Storage encryption at the level of the storage account with your own keys. When you specify a customer-managed key at the level of the storage account, that key is used to protect and control access the root encryption key for the storage account which in turn is used to encrypt and decrypt all blob and file data. Customer-managed keys offer greater flexibility to create, rotate, disable, and revoke access controls. You can also audit the encryption keys used to protect your data.

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?](#).

This 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 wrap and unwrap the account encryption key to perform encryption and decryption operations.

Enable customer-managed keys for a storage account

When you enable encryption with customer-managed keys for a storage account, Azure Storage wraps the account encryption key with the customer-managed key in the associated key vault. Enabling customer-managed keys does not impact performance, and the account is encrypted with the new key immediately, without any time delay.

A new storage account is always encrypted using Microsoft-managed keys. It's not possible to enable customer-managed keys at the time that the account is created. Customer-managed keys are stored in Azure Key Vault, and 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 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.

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 RSA keys of size 2048 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 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

To revoke access to customer-managed keys, use PowerShell or Azure CLI. For more information, see [Azure Key Vault PowerShell](#) or [Azure Key Vault CLI](#). Revoking access effectively blocks access to all data in the storage account, as the encryption key is inaccessible by Azure Storage.

Customer-managed keys for Azure managed disks (preview)

Customer-managed keys are also available for managing encryption of Azure managed disks (preview). 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.

Customer-provided keys (preview)

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.

For an example that shows how to specify a customer-provided key on a request to Blob storage, see [Specify a customer-provided key on a request to Blob storage with .NET](#).

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, 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
x-ms-encryption-key	Required for both write and read requests. A Base64-encoded AES-256 encryption key value.
x-ms-encryption-key-sha256	Required for both write and read requests. The Base64-encoded SHA256 of the encryption key.
x-ms-encryption-algorithm	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 passed on the request, download the blob and 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.

Azure Storage encryption versus disk encryption

Azure Storage encryption encrypts the page blobs that back Azure virtual machine disks. Additionally, all Azure virtual machine disks, including local temp disks, may optionally be encrypted with [Azure Disk Encryption](#). Azure

Disk Encryption uses industry-standard [BitLocker](#) on Windows and [DM-Crypt](#) on Linux to provide operating system-based encryption solutions that are integrated with Azure Key Vault.

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](#)

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](#).

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](#)

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 \(REST API\)](#).

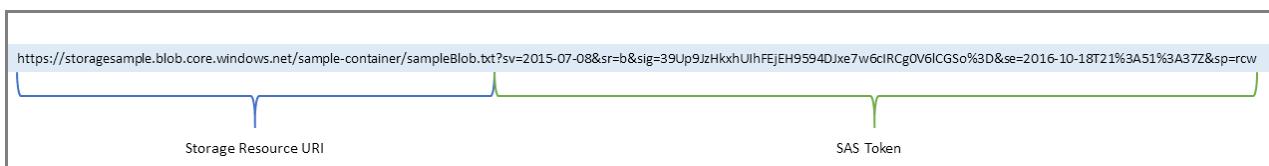
- With the storage account key. Both a service SAS and an account SAS are signed with the storage account key. To create a SAS that is signed with the account key, an application must have access to the account key.

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.

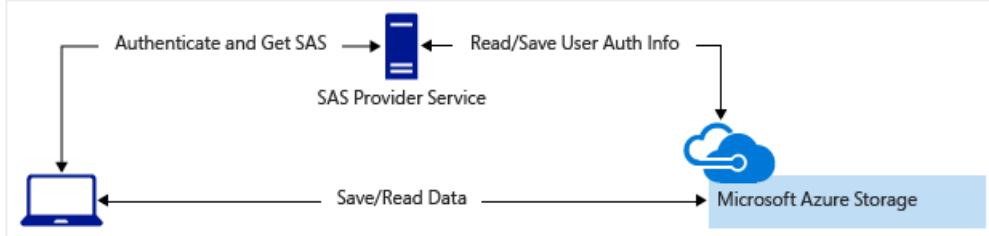
A common scenario where a SAS is useful is a service where users read and write their own data to your storage account. In a scenario where a storage account stores user data, there are two typical design patterns:

- 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 \(REST API\)](#)

Monitor, diagnose, and troubleshoot Microsoft Azure Storage

1/8/2020 • 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.

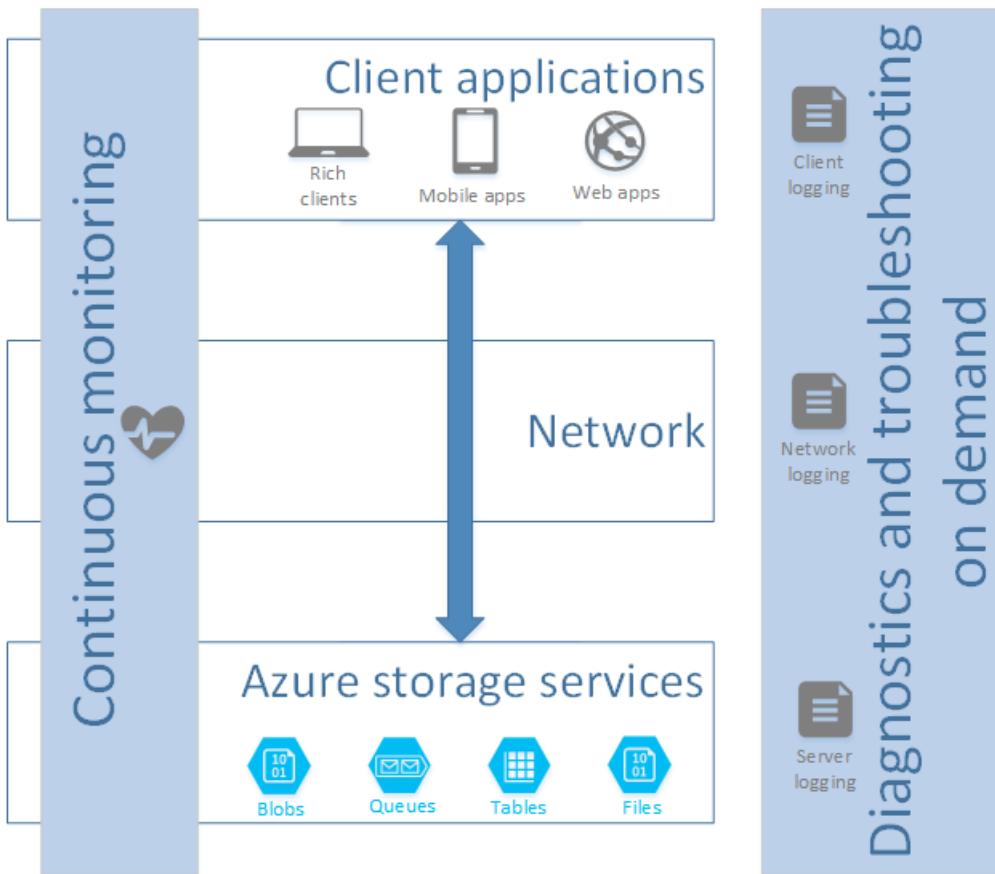
For a hands-on guide to end-to-end troubleshooting in Azure Storage applications, see [End-to-End Troubleshooting using Azure Storage Metrics and Logging, AzCopy, and Message Analyzer](#).

- [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.

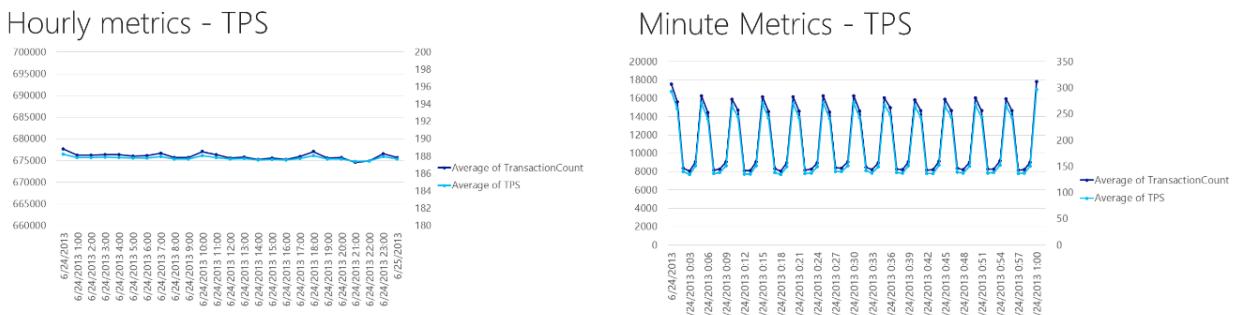
The storage service collects metrics using a best effort, but may not record every storage operation.

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.



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]

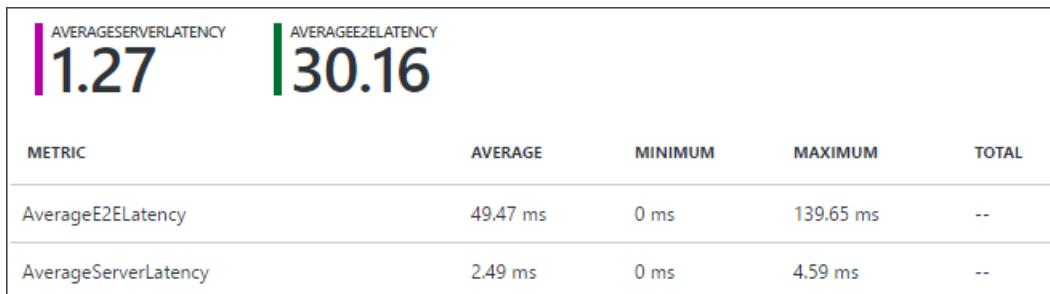
Your issue arises from using the storage emulator for development or test

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 show 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 https://domemaildist.blob.core.windows.net/azureimblobcontainer/blobCreatedViaSAS.txt?sv=2014-02-14&sr=c&si=mypolicy&sig=OFnd4Rd7z01fIvh%2BmcR6zbudIH2F5Ik%2FyhNYZEmJNQ%3D&api-version=2014-02-14
Microsoft.Azure.Storage	Information	3	85d077ab -...	Waiting for response.

Source	Verbosity	Verbosity	Client Request ID	Operation Text
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..
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...	DeleteIfExists method to delete the blob container. Note that this operation includes a HEAD request to check for the existence of the container.
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 https://domemaildist.blob.core.windows.net/azuremmblobcontainer .

REQUEST ID	OPERATION TEXT
07b26a5d-...	StringToSign = HEAD.....x-ms-client-request-id:07b26a5d-....x-ms-date:Tue, 03 Jun 2014 10:33:11 GMT.x-ms-version:2014-02-14./domemaildist/azuremmblobcontainer.restype:container.
07b26a5d-...	Waiting for response.
07b26a5d-...	Response received. Status code = 200, Request ID = eeead849-...Content-MD5 = , ETag = "0x8D14D2DC63D059B".
07b26a5d-...	Response headers were processed successfully, proceeding with the rest of the operation.
07b26a5d-...	Downloading response body.
07b26a5d-...	Operation completed successfully.
07b26a5d-...	Starting synchronous request to https://domemaildist.blob.core.windows.net/azuremmblobcontainer .
07b26a5d-...	StringToSign = DELETE.....x-ms-client-request-id:07b26a5d-....x-ms-date:Tue, 03 Jun 2014 10:33:12 GMT.x-ms-version:2014-02-14./domemaildist/azuremmblobcontainer.restype:container.
07b26a5d-...	Waiting for response.
07b26a5d-...	Response received. Status code = 202, Request ID = 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 https://domemaildist.blob.core.windows.net/azuremmblobcontainer .
e2d06d78-...	StringToSign = HEAD.....x-ms-client-request-id:e2d06d78-....x-ms-date:Tue, 03 Jun 2014 10:33:12 GMT.x-ms-version:2014-02-14./domemaildist/azuremmblobcontainer.restype:container.
e2d06d78-...	Waiting for response.
de8b1c3c-...	Starting synchronous request to https://domemaildist.blob.core.windows.net/azuremmblobcontainer/blobCreated.txt .

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 GMT.x-ms-version:2014-02-14./domemaildist/azuremmblobcontainer/blobCreated.txt.
de8b1c3c-...	Preparing to write request data.
e2d06d78-...	Exception thrown while waiting for response: The remote server returned an error: (404) Not Found..
e2d06d78-...	Response received. Status code = 404, Request ID = 353ae3bc-..., Content-MD5 = , ETag = .
e2d06d78-...	Response headers were processed successfully, proceeding with the rest of the operation.
e2d06d78-...	Downloading response body.
e2d06d78-...	Operation completed successfully.
e2d06d78-...	Starting asynchronous request to https://domemaildist.blob.core.windows.net/azuremmblobcontainer .
e2d06d78-...	StringToSign = PUT...0.....x-ms-client-request-id:e2d06d78-....x-ms-date:Tue, 03 Jun 2014 10:33:12 GMT.x-ms-version:2014-02-14./domemaildist/azuremmblobcontainer:restype:container.
e2d06d78-...	Waiting for response.
de8b1c3c-...	Writing request data.
de8b1c3c-...	Waiting for response.
e2d06d78-...	Exception thrown while waiting for response: The remote server returned an error: (409) Conflict..
e2d06d78-...	Response received. Status code = 409, Request ID = c27da20e-..., Content-MD5 = , ETag = .
e2d06d78-...	Downloading error response body.
de8b1c3c-...	Exception thrown while waiting for response: The remote server returned an error: (404) Not Found..
de8b1c3c-...	Response received. Status code = 404, Request ID = 0eaeb3e-..., Content-MD5 = , ETag = .
de8b1c3c-...	Exception thrown during the operation: The remote server returned an error: (404) Not Found..

REQUEST ID	OPERATION TEXT
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	https://domemaildist.blob.core.windows.net/azureimblobcontainer/blobCreatedViaSAS.txt
	?sv=2014-02-14&sr=c&si=mypolicy&sig=XXXXX&api-version=2014-02-14
Request ID header	a1f348d5-8032-4912-93ef-b393e5252a3b
Client request ID	2d064953-8436-4ee0-aa0c-65cb874f7929

Investigate why your client application is attempting to perform an operation for which it has not been granted permissions.

Client-side JavaScript code does not have permission to access the object

If you are using a JavaScript client and the storage service is returning HTTP 404 messages, you check for the following JavaScript errors in the browser:

```
SEC7120: Origin http://localhost:56309 not found in Access-Control-Allow-Origin header.
SCRIPT7002: XMLHttpRequest: Network Error 0x80070005, Access is denied.
```

NOTE

You can use the F12 Developer Tools in Internet Explorer to trace the messages exchanged between the browser and the storage service when you are troubleshooting client-side JavaScript issues.

These errors occur because the web browser implements the [same origin policy](#) security restriction that prevents a web page from calling an API in a different domain from the domain the page comes from.

To work around the JavaScript issue, you can configure Cross Origin Resource Sharing (CORS) for the storage service the client is accessing. For more information, see [Cross-Origin Resource Sharing \(CORS\) Support for Azure Storage Services](#).

The following code sample shows how to configure your blob service to allow JavaScript running in the Contoso domain to access a blob in your blob storage service:

```
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%\WASstorageEmulatorDb3*.*
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

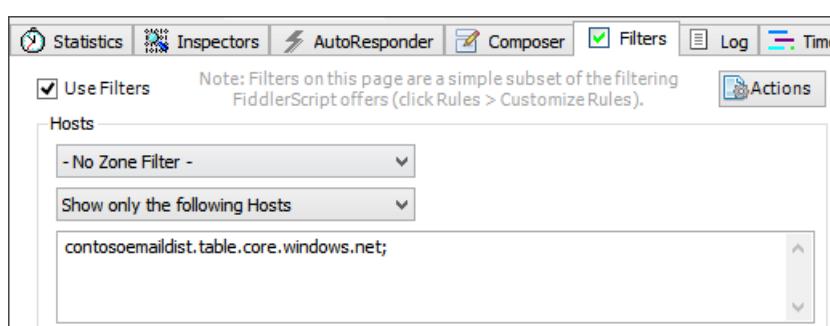
Fiddler can decode HTTPS traffic; you should read the Fiddler documentation carefully to understand how it does this, and to understand the security implications.

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:

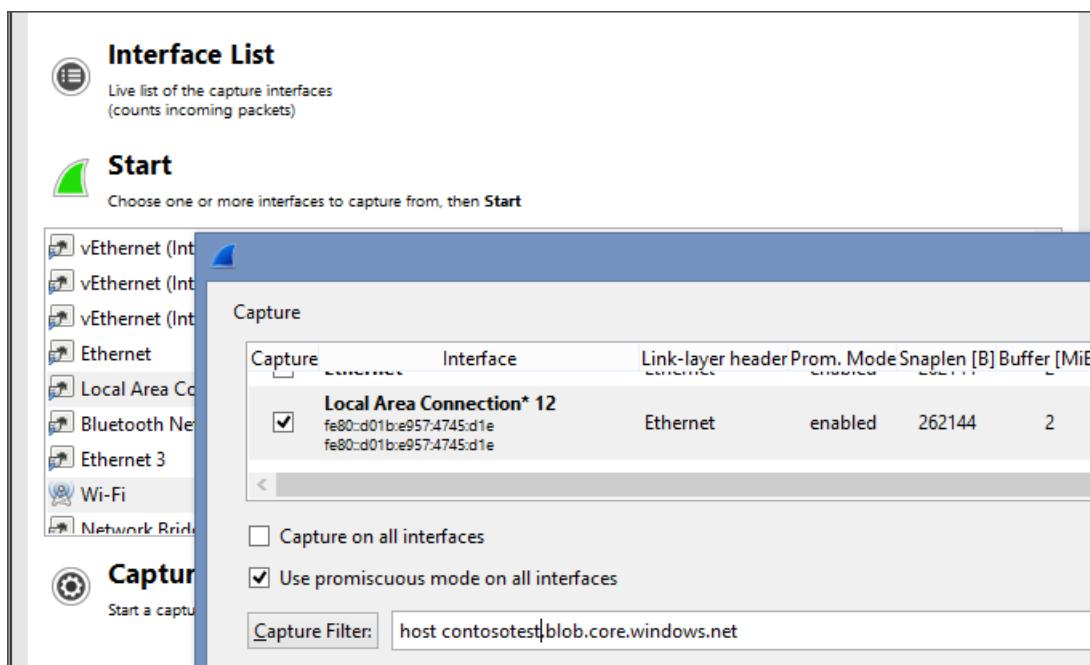


Appendix 2: Using Wireshark to capture network traffic

Wireshark is a network protocol analyzer that enables you to view detailed packet information for a wide range of network protocols.

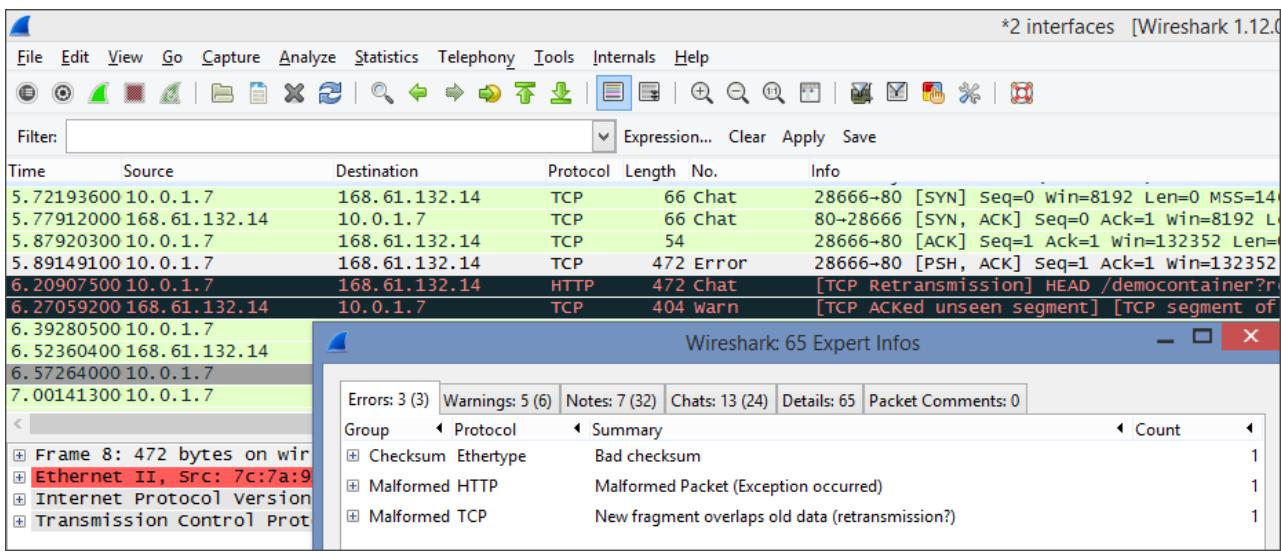
The following procedure shows you how to capture detailed packet information for traffic from the local machine where you installed Wireshark to the table service in your Azure storage account.

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](#).

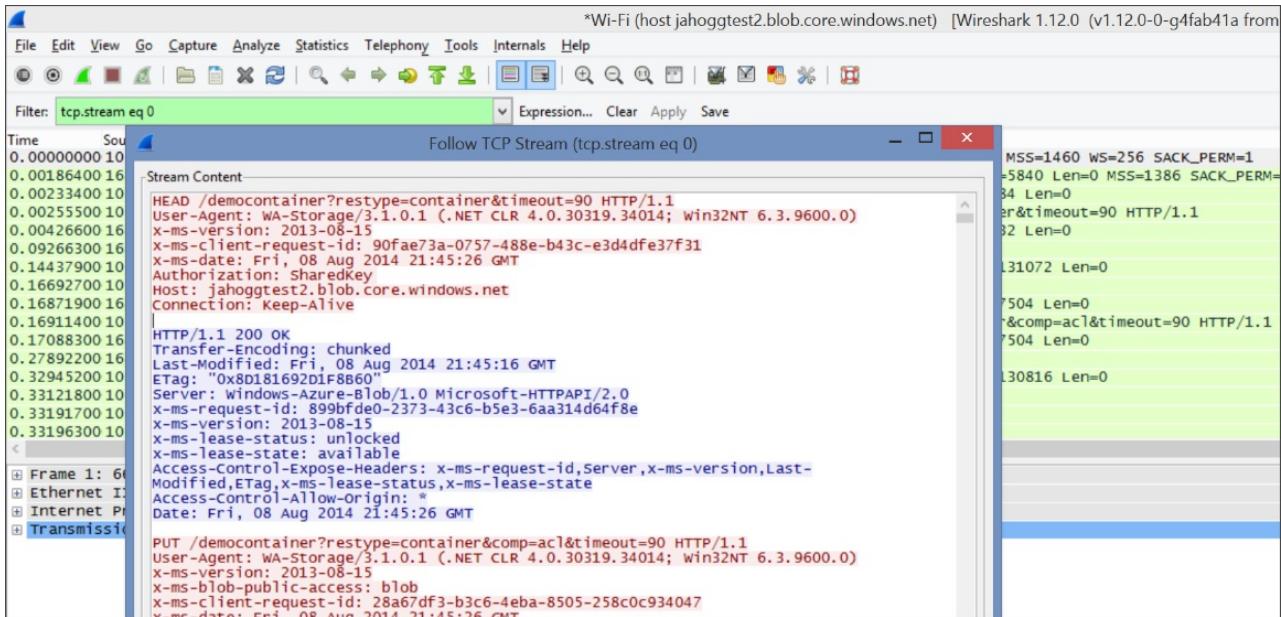


5. Click **Start**. Wireshark will now capture all the packets send to or from the table service endpoint as you use your client application on your local machine.
6. When you have finished, on the main menu click **Capture** and then **Stop**.
7. To save the captured data in a Wireshark Capture File, 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.



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:

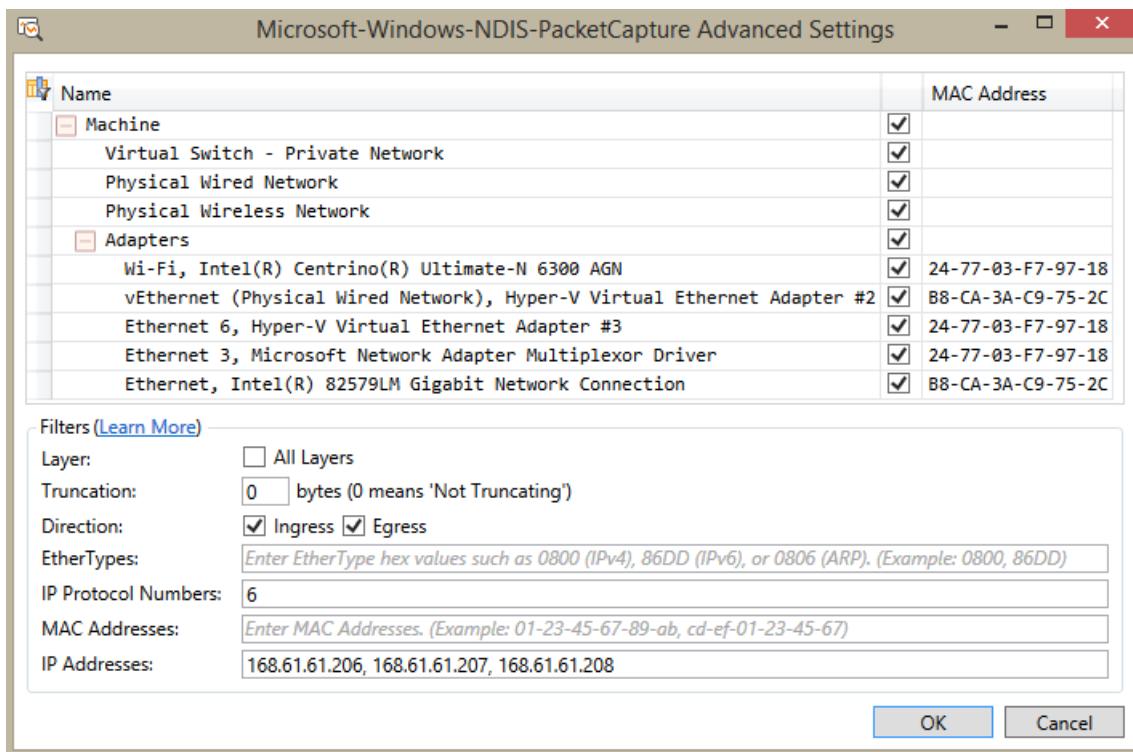
MessageNumber	Type	Timestamp	TimeElapsed	Source
302	i	2014-06-27T11:09:05.5912057	0	192.168.0.16
303	i	2014-06-27T11:09:05.6019881	0	192.168.0.16
304	i	2014-06-27T11:09:05.6020185	0	192.168.0.16
305	i	2014-06-27T11:09:05.6270940	0	168.61.61.207
306	i	2014-06-27T11:09:05.6270947	0.0001652	168.61.61.207
307	i	2014-06-27T11:09:05.6271621	0	168.61.61.207

Type Level Message

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-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

2/24/2020 • 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.

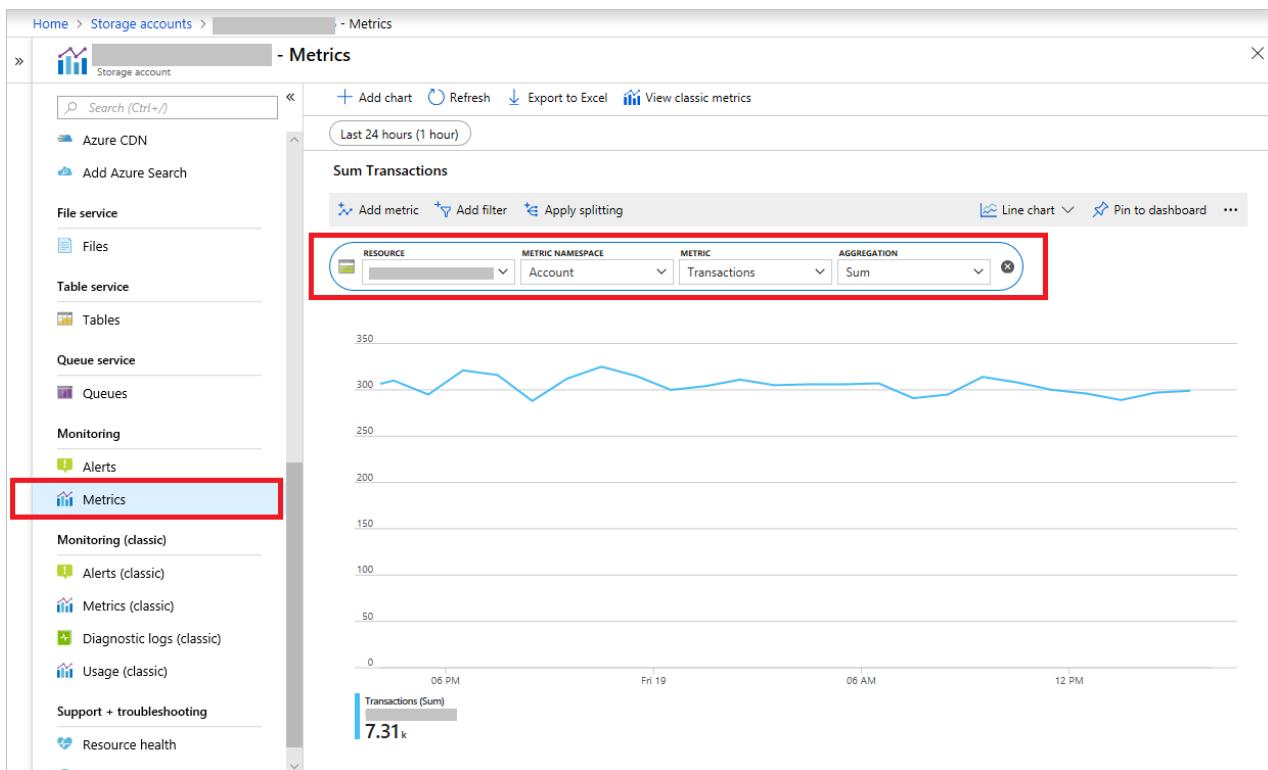
Access metrics

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](#).

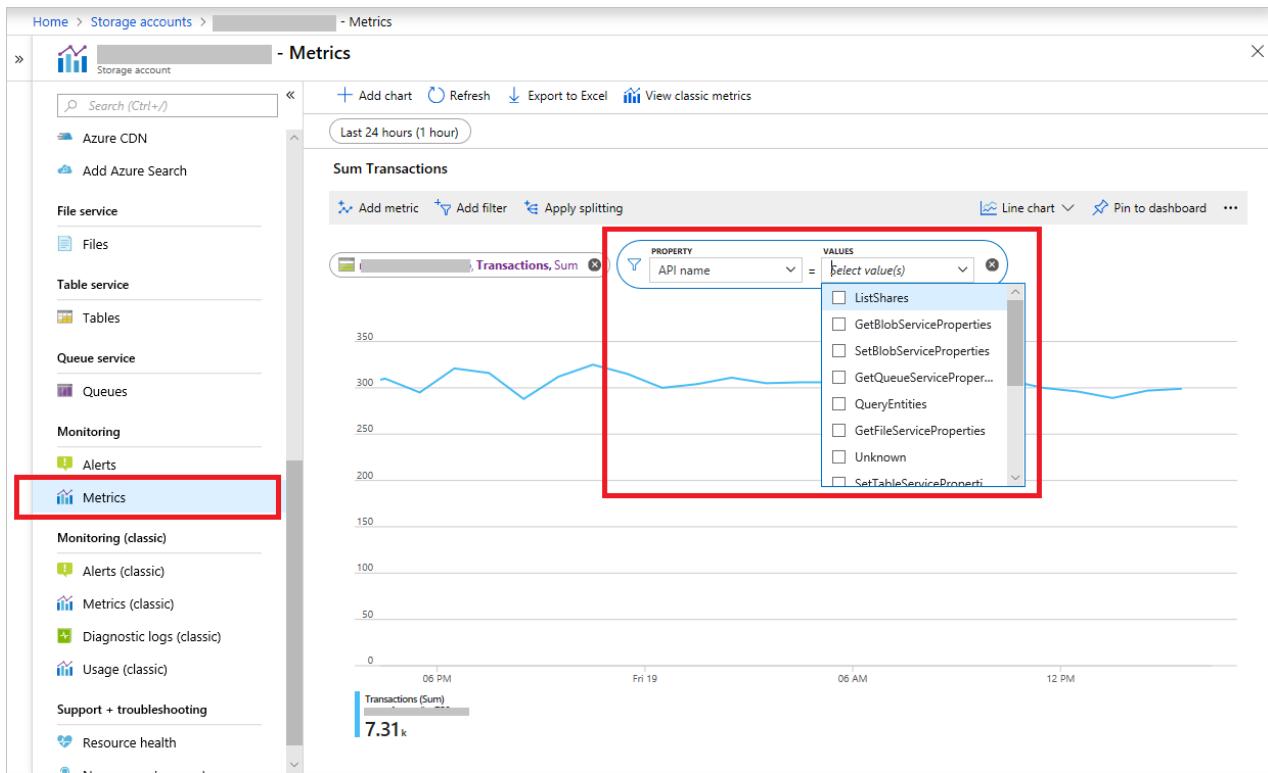
Metrics are enabled by default, and you can access the past 93 days of data. If you need to retain data for a longer period of time, you can archive metrics data to an Azure Storage account. This is configured in [diagnostic settings](#) in Azure Monitor.

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/storageAccount
s/{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/storageAccount
s/{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/storageAccount
s/{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}/{reso
urceType}/{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: BlobType, and BlobTier (Definition)</p>
BlobCount	<p>The number of blob objects stored in the storage account.</p> <p>Unit: Count Aggregation Type: Average Value example: 1024 Dimensions: BlobType, and BlobTier (Definition)</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 (Definition) 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 (Definition) Value example: 1024</p>
Egress	<p>The amount of egress data. This number includes egress from an external client into Azure Storage as well as egress within Azure. As a result, this number does not reflect billable egress.</p> <p>Unit: Bytes Aggregation Type: Total Applicable dimensions: GeoType, ApiName, and Authentication (Definition) 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 (Definition) 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 (Definition) 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 (Definition) Value example: 99.99</p>

Metrics dimensions

Azure Storage supports following dimensions for metrics in Azure Monitor.

DIMENSION NAME	DESCRIPTION
BlobType	The type of blob for Blob metrics only. The supported values are BlockBlob , PageBlob , and Azure Data Lake Storage . Append Blob is included in BlockBlob.
BlobTier	Azure storage offers different access tiers, which allow you to store blob object data in the most cost-effective manner. See more in Azure Storage blob tier . The supported values include: <ul style="list-style-type: none"> • Hot: Hot tier • Cool: Cool tier • Archive: Archive tier • Premium: Premium tier for block blob • P4/P6/P10/P15/P20/P30/P40/P50/P60: Tier types for premium page blob • Standard: Tier type for standard page Blob • Untiered: Tier type for general purpose v1 storage account
GeoType	Transaction from Primary or Secondary cluster. The available values include Primary and Secondary . It applies to Read Access Geo Redundant Storage(RA-GRS) when reading objects from secondary tenant.

DIMENSION NAME	DESCRIPTION
ResponseType	<p>Transaction response type. The available values include:</p> <ul style="list-style-type: none"> • ServerOtherError: All other server-side errors except described ones • ServerBusyError: Authenticated request that returned an HTTP 503 status code. • ServerTimeoutError: Timed-out authenticated request that returned an HTTP 500 status code. The timeout occurred due to a server error. • AuthorizationError: Authenticated request that failed due to unauthorized access of data or an authorization failure. • NetworkError: Authenticated request that failed due to network errors. Most commonly occurs when a client prematurely closes a connection before timeout expiration. • ClientAccountBandwidthThrottlingError: The request is throttled on bandwidth for exceeding storage account scalability limits. • ClientAccountRequestThrottlingError: The request is throttled on request rate for exceeding storage account scalability limits. • ClientThrottlingError: Other client-side throttling error. ClientAccountBandwidthThrottlingError and ClientAccountRequestThrottlingError are excluded. • ClientTimeoutError: 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. • ClientOtherError: All other client-side errors except described ones. • Success: Successful request • SuccessWithThrottling: Successful request when a SMB client gets throttled in the first attempt(s) but succeeds after retries.
ApiName	<p>The name of operation. For example:</p> <ul style="list-style-type: none"> • CreateContainer • DeleteBlob • GetBlob <p>For all operation names, see document.</p>
Authentication	<p>Authentication type used in transactions. The available values include:</p> <ul style="list-style-type: none"> • AccountKey: The transaction is authenticated with storage account key. • SAS: The transaction is authenticated with shared access signatures. • OAuth: The transaction is authenticated with OAuth access tokens. • Anonymous: The transaction is requested anonymously. It doesn't include preflight requests. • AnonymousPreflight: The transaction is preflight request.

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 BlobType equal to BlockBlob or PageBlob

OLD METRIC	NEW METRIC
ObjectCount	BlobCount with the dimension BlobType equal to BlockBlob or PageBlob
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
AnonymousAuthorizationError	Transactions with the dimension ResponseType equal to AuthorizationError and dimension Authentication equal to Anonymous
AnonymousClientOtherError	Transactions with the dimension ResponseType equal to ClientOtherError and dimension Authentication equal to Anonymous
AnonymousClientTimeoutError	Transactions with the dimension ResponseType equal to ClientTimeoutError and dimension Authentication equal to Anonymous
AnonymousNetworkError	Transactions with the dimension ResponseType equal to NetworkError and dimension Authentication equal to Anonymous
AnonymousServerOtherError	Transactions with the dimension ResponseType equal to ServerOtherError and dimension Authentication equal to Anonymous
AnonymousServerTimeoutError	Transactions with the dimension ResponseType equal to ServerTimeoutError and dimension Authentication equal to Anonymous
AnonymousSuccess	Transactions with the dimension ResponseType equal to Success and dimension Authentication equal to Anonymous
AnonymousThrottlingError	Transactions with the dimension ResponseType equal to ClientThrottlingError or ServerBusyError and dimension Authentication equal to Anonymous

OLD METRIC	NEW METRIC
AuthorizationError	Transactions with the dimension ResponseType equal to AuthorizationError
Availability	Availability
AverageE2ELatency	SuccessE2ELatency
AverageServerLatency	SuccessServerLatency
ClientOtherError	Transactions with the dimension ResponseType equal to ClientOtherError
ClientTimeoutError	Transactions with the dimension ResponseType equal to ClientTimeoutError
NetworkError	Transactions with the dimension ResponseType equal to NetworkError
PercentAuthorizationError	Transactions with the dimension ResponseType equal to AuthorizationError
PercentClientOtherError	Transactions with the dimension ResponseType equal to ClientOtherError
PercentNetworkError	Transactions with the dimension ResponseType equal to NetworkError
PercentServerOtherError	Transactions with the dimension ResponseType equal to ServerOtherError
PercentSuccess	Transactions with the dimension ResponseType equal to Success
PercentThrottlingError	Transactions with the dimension ResponseType equal to ClientThrottlingError or ServerBusyError
PercentTimeoutError	Transactions with the dimension ResponseType equal to ServerTimeoutError or ResponseType equal to ClientTimeoutError
SASAuthorizationError	Transactions with the dimension ResponseType equal to AuthorizationError and dimension Authentication equal to SAS
SASClientOtherError	Transactions with the dimension ResponseType equal to ClientOtherError and dimension Authentication equal to SAS
SASClientTimeoutError	Transactions with the dimension ResponseType equal to ClientTimeoutError and dimension Authentication equal to SAS
SASNetworkError	Transactions with the dimension ResponseType equal to NetworkError and dimension Authentication equal to SAS

OLD METRIC	NEW METRIC
SASServerOtherError	Transactions with the dimension ResponseType equal to ServerOtherError and dimension Authentication equal to SAS
SASServerTimeoutError	Transactions with the dimension ResponseType equal to ServerTimeoutError and dimension Authentication equal to SAS
SASSuccess	Transactions with the dimension ResponseType equal to Success and dimension Authentication equal to SAS
SASThrottlingError	Transactions with the dimension ResponseType equal to ClientThrottlingError or ServerBusyError and dimension Authentication equal to SAS
ServerOtherError	Transactions with the dimension ResponseType equal to ServerOtherError
ServerTimeoutError	Transactions with the dimension ResponseType equal to ServerTimeoutError
Success	Transactions with the dimension ResponseType equal to Success
ThrottlingError	Transactions with the dimension ResponseType equal to ClientThrottlingError or ServerBusyError
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 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	TotalBillerableRequests	TotalIngress	TotalEgress	Availability	Average2ELatency	AverageServerLatency	PercentSuccess
20140522T1100	user;All	2014-05-22T11:01:16.7650250Z	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.ToUniversalTime().ToString("yyyyMMdd'T'HHmm");
    var end = endTime.ToUniversalTime().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](#)

Frequently asked questions (FAQ) about Azure Files

2/25/2020 • 28 minutes to read • [Edit Online](#)

Azure Files offers fully managed file shares in the cloud that are accessible via the industry-standard [Server Message Block \(SMB\) protocol](#). You can mount Azure file shares concurrently on cloud or on-premises deployments of Windows, Linux, and macOS. You also can cache Azure file shares on Windows Server machines by using Azure File Sync for fast access close to where the data is used.

This article answers common questions about Azure Files features and functionality, including the use of Azure File Sync with Azure Files. If you don't see the answer to your question, you can contact us through the following channels (in escalating order):

1. The comments section of this article.
2. [Azure Storage Forum](#).
3. [Azure Files UserVoice](#).
4. Microsoft Support. To create a new support request, in the Azure portal, on the **Help** tab, select the **Help + support** button, and then select **New support request**.

General

- **How is Azure Files useful?**

You can use Azure Files to create file shares in the cloud, without being responsible for managing the overhead of a physical server, device, or appliance. We do the monotonous work for you, including applying OS updates and replacing bad disks. To learn more about the scenarios that Azure Files can help you with, see [Why Azure Files is useful](#).

- **What are different ways to access files in Azure Files?**

You can mount the file share on your local machine by using the SMB 3.0 protocol, or you can use tools like [Storage Explorer](#) to access files in your file share. From your application, you can use storage client libraries, REST APIs, PowerShell, or Azure CLI to access your files in the Azure file share.

- **What is Azure File Sync?**

You can use Azure File Sync to centralize your organization's file shares in Azure Files, while keeping the flexibility, performance, and compatibility of an on-premises file server. Azure File Sync transforms your Windows Server machines into a quick cache of your Azure file share. You can use any protocol that's available on Windows Server to access your data locally, including SMB, Network File System (NFS), and File Transfer Protocol Service (FTPS). You can have as many caches as you need across the world.

- **Why would I use an Azure file share versus Azure Blob storage for my data?**

Azure Files and Azure Blob storage both offer ways to store large amounts of data in the cloud, but they are useful for slightly different purposes.

Azure Blob storage is useful for massive-scale, cloud-native applications that need to store unstructured data. To maximize performance and scale, Azure Blob storage is a simpler storage abstraction than a true file system. You can access Azure Blob storage only through REST-based client libraries (or directly through the REST-based protocol).

Azure Files is specifically a file system. Azure Files has all the file abstractions that you know and love from years of working with on-premises operating systems. Like Azure Blob storage, Azure Files offers a REST interface and REST-based client libraries. Unlike Azure Blob storage, Azure Files offers SMB access to Azure file shares. By using SMB, you can mount an Azure file share directly on Windows, Linux, or macOS,

either on-premises or in cloud VMs, without writing any code or attaching any special drivers to the file system. You also can cache Azure file shares on on-premises file servers by using Azure File Sync for quick access, close to where the data is used.

For a more in-depth description on the differences between Azure Files and Azure Blob storage, see [Deciding when to use Azure Blob storage, Azure Files, or Azure Disks](#). To learn more about Azure Blob storage, see [Introduction to Blob storage](#).

- **Why would I use an Azure file share instead of Azure Disks?**

A disk in Azure Disks is simply a disk. To get value from Azure Disks, you must attach a disk to a virtual machine that's running in Azure. Azure Disks can be used for everything that you would use a disk for on an on-premises server. You can use it as an OS system disk, as swap space for an OS, or as dedicated storage for an application. An interesting use for Azure Disks is to create a file server in the cloud to use in the same places where you might use an Azure file share. Deploying a file server in Azure Virtual Machines is a high-performance way to get file storage in Azure when you require deployment options that currently are not supported by Azure Files (such as NFS protocol support or premium storage).

However, running a file server with Azure Disks as back-end storage typically is much more expensive than using an Azure file share, for a few reasons. First, in addition to paying for disk storage, you also must pay for the expense of running one or more Azure VMs. Second, you also must manage the VMs that are used to run the file server. For example, you are responsible for OS upgrades. Finally, if you ultimately require data to be cached on-premises, it's up to you to set up and manage replication technologies, such as Distributed File System Replication (DFSR), to make that happen.

One approach to getting the best of both Azure Files and a file server that's hosted in Azure Virtual Machines (in addition to using Azure Disks as back-end storage) is to install Azure File Sync on a file server that's hosted on a cloud VM. If the Azure file share is in the same region as your file server, you can enable cloud tiering and set the volume of free space percentage to maximum (99%). This ensures minimal duplication of data. You also can use any applications you want with your file servers, like applications that require NFS protocol support.

For information about an option for setting up a high-performance and highly available file server in Azure, see [Deploying IaaS VM guest clusters in Microsoft Azure](#). For a more in-depth description of the differences between Azure Files and Azure Disks, see [Deciding when to use Azure Blob storage, Azure Files, or Azure Disks](#). To learn more about Azure Disks, see [Azure Managed Disks overview](#).

- **How do I get started using Azure Files?**

Getting started with Azure Files is easy. First, [create a file share](#), and then mount it in your preferred operating system:

- [Mount in Windows](#)
- [Mount in Linux](#)
- [Mount in macOS](#)

For a more in-depth guide about deploying an Azure file share to replace production file shares in your organization, see [Planning for an Azure Files deployment](#).

- **What storage redundancy options are supported by Azure Files?**

Currently, Azure Files supports locally redundant storage (LRS), zone redundant storage (ZRS), geo-redundant storage (GRS), and geo-zone-redundant storage (GZRS) (preview). We plan to support read-access geo-redundant (RA-GRS) storage in the future, but we don't have timelines to share at this time.

- **What storage tiers are supported in Azure Files?**

Azure Files supports two storage tiers: premium and standard. Standard file shares are created in general purpose (GPv1 or GPv2) storage accounts and premium file shares are created in FileStorage storage

accounts. Learn more about how to create [standard file shares](#) and [premium file shares](#).

NOTE

You cannot create Azure file shares from Blob storage accounts or *premium* general purpose (GPv1 or GPv2) storage accounts. Standard Azure file shares must be created in *standard* general purpose accounts only and premium Azure file shares must be created in FileStorage storage accounts only. *Premium* general purpose (GPv1 and GPv2) storage accounts are for premium page blobs only.

- **I really want to see a specific feature added to Azure Files. Can you add it?**

The Azure Files team is interested in hearing any and all feedback you have about our service. Please vote on feature requests at [Azure Files UserVoice](#)! We're looking forward to delighting you with many new features.

Azure File Sync

- **What regions are supported for Azure File Sync?**

The list of available regions can be found on the [Region availability](#) section of the Azure File Sync planning guide. We will continuously add support for additional regions, including non-Public regions.

- **Can I have domain-joined and non-domain-joined servers in the same sync group?**

Yes. A sync group can contain server endpoints that have different Active Directory memberships, even if they are not domain-joined. Although this configuration technically works, we do not recommend this as a typical configuration because access control lists (ACLs) that are defined for files and folders on one server might not be able to be enforced by other servers in the sync group. For best results, we recommend syncing between servers that are in the same Active Directory forest, between servers that are in different Active Directory forests but which have established trust relationships, or between servers that are not in a domain. We recommend that you avoid using a mix of these configurations.

- **I created a file directly in my Azure file share by using SMB or in the portal. How long does it take for the file to sync to the servers in the sync group?**

Changes made to the Azure file share by using the Azure portal or SMB are not immediately detected and replicated like changes to the server endpoint. Azure Files does not yet have change notifications or journaling, so there's no way to automatically initiate a sync session when files are changed. On Windows Server, Azure File Sync uses [Windows USN journaling](#) to automatically initiate a sync session when files change.

To detect changes to the Azure file share, Azure File Sync has a scheduled job called a *change detection job*. A change detection job enumerates every file in the file share, and then compares it to the sync version for that file. When the change detection job determines that files have changed, Azure File Sync initiates a sync session. The change detection job is initiated every 24 hours. Because the change detection job works by enumerating every file in the Azure file share, change detection takes longer in larger namespaces than in smaller namespaces. For large namespaces, it might take longer than once every 24 hours to determine which files have changed.

To immediately sync files that are changed in the Azure file share, the **Invoke-AzStorageSyncChangeDetection** PowerShell cmdlet can be used to manually initiate the detection of changes in the Azure file share. This cmdlet is intended for scenarios where some type of automated process is making changes in the Azure file share or the changes are done by an administrator (like moving files and directories into the share). For end user changes, the recommendation is to install the Azure File Sync agent in an IaaS VM and have end users access the file share through the IaaS VM. This way all changes will quickly sync to other agents without the need to use the **Invoke-AzStorageSyncChangeDetection** cmdlet. To learn more, see the [Invoke-AzStorageSyncChangeDetection](#)

documentation.

NOTE

Changes made to an Azure file share using REST does not update the SMB last modified time and will not be seen as a change by sync.

We are exploring adding change detection for an Azure file share similar to USN for volumes on Windows Server. Help us prioritize this feature for future development by voting for it at [Azure Files UserVoice](#).

- **If the same file is changed on two servers at approximately the same time, what happens?**

Azure File Sync uses a simple conflict-resolution strategy: we keep both changes to files that are changed on two servers at the same time. The most recently written change keeps the original file name. The older file has the "source" machine and the conflict number appended to the name. It follows this taxonomy:

<FileNameWithoutExtension>-<MachineName>[-#].<ext>

For example, the first conflict of CompanyReport.docx would become CompanyReport-CentralServer.docx if CentralServer is where the older write occurred. The second conflict would be named CompanyReport-CentralServer-1.docx. Azure File Sync supports 100 conflict files per file. Once the maximum number of conflict files has been reached, the file will fail to sync until the number of conflict files is less than 100.

- **Is geo-redundant storage supported for Azure File Sync?**

Yes, Azure Files supports both locally redundant storage (LRS) and geo-redundant storage (GRS). If you initiate a storage account failover between paired regions from an account configured for GRS, Microsoft recommends that you treat the new region as a backup of data only. Azure File Sync does not automatically begin syncing with the new primary region.

- **Why doesn't the *Size on disk* property for a file match the *Size* property after using Azure File Sync?**

See [Understanding Cloud Tiering](#).

- **How can I tell whether a file has been tiered?**

See [Understanding Cloud Tiering](#).

- **A file I want to use has been tiered. How can I recall the file to disk to use it locally?**

See [Understanding Cloud Tiering](#).

- **How do I force a file or directory to be tiered?**

See [Understanding Cloud Tiering](#).

- **How is *volume free space* interpreted when I have multiple server endpoints on a volume?**

See [Understanding Cloud Tiering](#).

- **Which files or folders are automatically excluded by Azure File Sync?**

See [Files skipped](#).

- **Can I use Azure File Sync with either Windows Server 2008 R2, Linux, or my network-attached storage (NAS) device?**

Currently, Azure File Sync supports only Windows Server 2019, Windows Server 2016, and Windows Server 2012 R2. At this time, we don't have any other plans we can share, but we're open to supporting additional platforms based on customer demand. Let us know at [Azure Files UserVoice](#) what platforms you would like us to support.

- **Why do tiered files exist outside of the server endpoint namespace?**

Prior to Azure File Sync agent version 3, Azure File Sync blocked the move of tiered files outside the server endpoint but on the same volume as the server endpoint. Copy operations, moves of non-tiered files, and

moves of tiered to other volumes were unaffected. The reason for this behavior was the implicit assumption that File Explorer and other Windows APIs have that move operations on the same volume are (nearly) instantaneous rename operations. This means moves will make File Explorer or other move methods (such as command line or PowerShell) appear unresponsive while Azure File Sync recalls the data from the cloud. Starting with [Azure File Sync agent version 3.0.12.0](#), Azure File Sync will allow you to move a tiered file outside of the server endpoint. We avoid the negative effects previously mentioned by allowing the tiered file to exist as a tiered file outside of the server endpoint and then recalling the file in the background. This means that moves on the same volume are instantaneous, and we do all the work to recall the file to disk after the move has completed.

- **I'm having an issue with Azure File Sync on my server (sync, cloud tiering, etc.). Should I remove and recreate my server endpoint?**

No: removing a server endpoint isn't like rebooting a server! Removing and recreating the server endpoint is almost never an appropriate solution to fixing issues with sync, cloud tiering, or other aspects of Azure File Sync. Removing a server endpoint is a destructive operation. It may result in data loss in the case that tiered files exist outside of the server endpoint namespace. See [Why do tiered files exist outside of the server endpoint namespace](#) for more information. Or it may result in inaccessible files for tiered files that exist within the server endpoint namespace. These issues won't resolve when the server endpoint is recreated. Tiered files may exist within your server endpoint namespace even if you never had cloud tiering enabled. That's why recommend that you don't remove the server endpoint unless you would like to stop using Azure File Sync with this particular folder or have been explicitly instructed to do so by a Microsoft engineer. For more information on remove server endpoints, see [Remove a server endpoint](#).

- **Can I move the storage sync service and/or storage account to a different resource group or subscription?**

Yes, the storage sync service and/or storage account can be moved to a different resource group or subscription within the existing Azure AD tenant. If the storage account is moved, you need to give the Hybrid File Sync Service access to the storage account (see [Ensure Azure File Sync has access to the storage account](#)).

NOTE

Azure File Sync does not support moving the subscription to a different Azure AD tenant.

- **Does Azure File Sync preserve directory/file level NTFS ACLs along with data stored in Azure Files?**

As of February 24th, 2020, new and existing ACLs tiered by Azure file sync will be persisted in NTFS format, and ACL modifications made directly to the Azure file share will sync to all servers in the sync group. Any changes on ACLs made to Azure Files will sync down via Azure file sync. When copying data to Azure Files, make sure you use SMB to access the share and preserve your ACLs. Existing REST based tools, such as AzCopy or Storage Explorer do not persist ACLs.

If you have enabled Azure Backup on your file sync managed file shares, file ACLs can continue to be restored as part of the backup restore workflow. This works either for the entire share or individual files/directories.

If you are using snapshots as part of the self-managed backup solution for file shares managed by file sync, your ACLs may not be restored properly to NTFS ACLs if the snapshots were taken prior to February 24th, 2020. If this occurs, consider contacting Azure Support.

Security, authentication, and access control

- **Is identity-based authentication and access control supported by Azure Files?**

Yes, Azure Files supports identity-based authentication and access control. You can choose one of two ways to use identity-based access control: Active Directory (AD) (preview) or Azure Active Directory Domain Services (Azure AD DS) (GA). AD supports authentication using AD domain joined machines, either on-premises or in Azure, to access Azure file shares over SMB. Azure AD DS authentication over SMB for Azure Files enables Azure AD DS domain-joined Windows VMs to access shares, directories, and files using Azure AD credentials. For more details, see [Overview of Azure Files identity-based authentication support for SMB access](#).

Azure Files offers two additional ways to manage access control:

- You can use shared access signatures (SAS) to generate tokens that have specific permissions, and which are valid for a specified time interval. For example, you can generate a token with read-only access to a specific file that has a 10-minute expiry. Anyone who possesses the token while the token is valid has read-only access to that file for those 10 minutes. Shared access signature keys are supported only via the REST API or in client libraries. You must mount the Azure file share over SMB by using the storage account keys.
- Azure File Sync preserves and replicates all discretionary ACLs, or DACLs, (whether Active Directory-based or local) to all server endpoints that it syncs to. Because Windows Server can already authenticate with Active Directory, Azure File Sync is an effective stop-gap option until full support for Active Directory-based authentication and ACL support arrives.

You can refer to [Authorizing access to Azure Storage](#) for a comprehensive representation of all protocols supported on Azure Storage services.

- **Does Azure Files Azure Active Directory Domain Services (Azure AD DS) Authentication support SMB access using Azure AD credentials from devices joined to or registered with Azure AD?**

No, this scenario is not supported.

- **Are there REST APIs to support Get/Set/Copy directory/file NTFS ACLs?**

Yes, we support REST APIs that get, set, or copy NTFS ACLs for directories or files when using the [2019-02-02](#) (or later) REST API.

- **Can I access Azure Files with Azure AD credentials from a VM under a different subscription?**

If the subscription under which the file share is deployed is associated with the same Azure AD tenant as the Azure AD Domain Services deployment to which the VM is domain-joined, then you can then access Azure Files using the same Azure AD credentials. The limitation is imposed not on the subscription but on the associated Azure AD tenant.

- **Can I enable either Azure Files Azure AD DS or AD authentication with an Azure AD tenant that is different from the primary tenant which the file share is associated with?**

No, Azure Files only supports Azure AD DS or AD integration with an Azure AD tenant that resides in the same subscription as the file share. Only one subscription can be associated with an Azure AD tenant. This limitation applies to both Azure AD DS and AD authentication methods. When using AD for authentication, the AD credential must be synced to the Azure AD that the storage account is associated with.

- **Does Azure Files Azure AD DS or AD authentication support Linux VMs?**

No, authentication from Linux VMs is not supported.

- **Does Azure Files AD authentication support integration with an AD environment using multiple forests?**

Azure Files AD authentication only integrates with the forest of the AD domain service that the storage account is registered to. To support authentication from another AD forest, your environment must have

forest trust configured properly. The way Azure Files register to an AD domain service is mostly the same as a regular file server, where it creates an identity (computer or service logon account) in AD for authentication. The only difference is that the registered SPN of the storage account ends with "file.core.windows.net" which does not match with the domain suffix. Consult your domain administrator to see if any update to your DNS routing policy is required to enable multiple forest authentication due to the different domain suffix.

- **What regions are available for Azure Files AD authentication (preview)?**

Refer to [AD regional availability](#) for details.

- **Can I leverage Azure Files Azure AD DS authentication or Active Directory (AD) authentication (preview) on file shares managed by Azure File Sync?**

Yes, you can enable Azure AD DS or AD authentication on a file share managed by Azure file sync. Changes to the directory/file NTFS ACLs on local file servers will be tiered to Azure Files and vice-versa.

- **How can I ensure that my Azure file share is encrypted at rest?**

Yes. For more information see [Azure Storage Service Encryption](#).

- **How can I provide access to a specific file by using a web browser?**

You can use shared access signatures to generate tokens that have specific permissions, and which are valid for a specified time interval. For example, you can generate a token that gives read-only access to a specific file, for a set period of time. Anyone who possesses the URL can access the file directly from any web browser while the token is valid. You can easily generate a shared access signature key from a UI like Storage Explorer.

- **Is it possible to specify read-only or write-only permissions on folders within the share?**

If you mount the file share by using SMB, you don't have folder-level control over permissions. However, if you create a shared access signature by using the REST API or client libraries, you can specify read-only or write-only permissions on folders within the share.

- **Can I implement IP restrictions for an Azure file share?**

Yes. Access to your Azure file share can be restricted at the storage account level. For more information, see [Configure Azure Storage Firewalls and Virtual Networks](#).

- **What data compliance policies does Azure Files support?**

Azure Files runs on top of the same storage architecture that's used in other storage services in Azure Storage. Azure Files applies the same data compliance policies that are used in other Azure storage services. For more information about Azure Storage data compliance, you can refer to [Azure Storage compliance offerings](#), and go to the [Microsoft Trust Center](#).

On-premises access

- **My ISP or IT blocks Port 445 which is failing Azure Files mount. What should I do?**

You can learn about [various ways to workaround blocked port 445 here](#). Azure Files only allows connections using SMB 3.0 (with encryption support) from outside the region or datacenter. SMB 3.0 protocol has introduced many security features including channel encryption which is very secure to use over internet. However its possible that port 445 has been blocked due to historical reasons of vulnerabilities found in lower SMB versions. In ideal case, the port should be blocked for only for SMB 1.0 traffic and SMB 1.0 should be turned off on all clients.

- **Do I have to use Azure ExpressRoute to connect to Azure Files or to use Azure File Sync on-**

premises?

No. ExpressRoute is not required to access an Azure file share. If you are mounting an Azure file share directly on-premises, all that's required is to have port 445 (TCP outbound) open for internet access (this is the port that SMB uses to communicate). If you're using Azure File Sync, all that's required is port 443 (TCP outbound) for HTTPS access (no SMB required). However, you *can* use ExpressRoute with either of these access options.

- **How can I mount an Azure file share on my local machine?**

You can mount the file share by using the SMB protocol if port 445 (TCP outbound) is open and your client supports the SMB 3.0 protocol (for example, if you're using Windows 10 or Windows Server 2016). If port 445 is blocked by your organization's policy or by your ISP, you can use Azure File Sync to access your Azure file share.

Backup

- **How do I back up my Azure file share?**

You can use periodic [share snapshots](#) for protection against accidental deletions. You also can use AzCopy, Robocopy, or a third-party backup tool that can back up a mounted file share. Azure Backup offers backup of Azure Files. Learn more about [back up Azure file shares by Azure Backup](#).

Share snapshots

Share snapshots: General

- **What are file share snapshots?**

You can use Azure file share snapshots to create a read-only version of your file shares. You also can use Azure Files to copy an earlier version of your content back to the same share, to an alternate location in Azure, or on-premises for more modifications. To learn more about share snapshots, see the [Share snapshot overview](#).

- **Where are my share snapshots stored?**

Share snapshots are stored in the same storage account as the file share.

- **Are share snapshots application-consistent?**

No, share snapshots are not application-consistent. The user must flush the writes from the application to the share before taking the share snapshot.

- **Are there limits on the number of share snapshots I can use?**

Yes. Azure Files can retain a maximum of 200 share snapshots. Share snapshots do not count toward the share quota, so there is no per-share limit on the total space that's used by all the share snapshots. Storage account limits still apply. After 200 share snapshots, you must delete older snapshots to create new share snapshots.

- **How much do share snapshots cost?**

Standard transaction and standard storage cost will apply to snapshot. Snapshots are incremental in nature. The base snapshot is the share itself. All the subsequent snapshots are incremental and will only store the diff from the previous snapshot. This means that the delta changes that will be seen in the bill will be minimal if your workload churn is minimal. See [Pricing page](#) for Standard Azure Files pricing information. Today the way to look at size consumed by share snapshot is by comparing the billed capacity with used capacity. We are working on tooling to improve the reporting.

- **Are NTFS ACLs on directories and files persisted in share snapshots?**

NTFS ACLs on directories and files are persisted in share snapshots.

[Create share snapshots](#)

- **Can I create share snapshot of individual files?**

Share snapshots are created at the file share level. You can restore individual files from the file share snapshot, but you cannot create file-level share snapshots. However, if you have taken a share-level share snapshot and you want to list share snapshots where a specific file has changed, you can do this under **Previous Versions** on a Windows-mounted share.

If you need a file snapshot feature, let us know at [Azure Files UserVoice](#).

- **Can I create share snapshots of an encrypted file share?**

You can take a share snapshot of Azure file shares that have encryption at rest enabled. You can restore files from a share snapshot to an encrypted file share. If your share is encrypted, your share snapshot also is encrypted.

- **Are my share snapshots geo-redundant?**

Share snapshots have the same redundancy as the Azure file share for which they were taken. If you have selected geo-redundant storage for your account, your share snapshot also is stored redundantly in the paired region.

Manage share snapshots

- **Can I browse my share snapshots from Linux?**

You can use Azure CLI to create, list, browse, and restore share snapshots in Linux.

- **Can I copy the share snapshots to a different storage account?**

You can copy files from share snapshots to another location, but you cannot copy the share snapshots themselves.

Restore data from share snapshots

- **Can I promote a share snapshot to the base share?**

You can copy data from a share snapshot to any other destination. You cannot promote a share snapshot to the base share.

- **Can I restore data from my share snapshot to a different storage account?**

Yes. Files from a share snapshot can be copied to the original location or to an alternate location that includes either the same storage account or a different storage account, in either the same region or in different regions. You also can copy files to an on-premises location or to any other cloud.

Clean up share snapshots

- **Can I delete my share but not delete my share snapshots?**

If you have active share snapshots on your share, you cannot delete your share. You can use an API to delete share snapshots, along with the share. You also can delete both the share snapshots and the share in the Azure portal.

- **What happens to my share snapshots if I delete my storage account?**

If you delete your storage account, the share snapshots also are deleted.

Billing and pricing

- **Does the network traffic between an Azure VM and an Azure file share count as external bandwidth that is charged to the subscription?**

If the file share and VM are in the same Azure region, there is no additional charge for the traffic between the file share and the VM. If the file share and the VM are in different regions, the traffic between them are charged as external bandwidth.

- **How much do share snapshots cost?**

During preview, there is no charge for share snapshot capacity. Standard storage egress and transaction costs apply. After general availability, subscriptions will be charged for capacity and transactions on share

snapshots.

Share snapshots are incremental in nature. The base share snapshot is the share itself. All subsequent share snapshots are incremental and store only the difference from the preceding share snapshot. You are billed only for the changed content. If you have a share with 100 GiB of data but only 5 GiB has changed since your last share snapshot, the share snapshot consumes only 5 additional GiB, and you are billed for 105 GiB. For more information about transaction and standard egress charges, see the [Pricing page](#).

Scale and performance

- **What are the scale limits of Azure Files?**

For information about scalability and performance targets for Azure Files, see [Azure Files scalability and performance targets](#).

- **What sizes are available for Azure file shares?**

Azure file share sizes (premium and standard) can scale up to 100 TiB. See the [Onboard to larger file shares \(standard tier\)](#) section of the planning guide for onboarding instructions to the larger file shares for the standard tier.

- **Does expanding my file share quota impact my workloads or Azure File Sync?**

No. Expanding the quota will not impact your workloads or Azure File Sync.

- **How many clients can access the same file simultaneously?**

There is a quota of 2,000 open handles on a single file. When you have 2,000 open handles, an error message is displayed that says the quota is reached.

- **My performance is slow when I unzip files in Azure Files. What should I do?**

To transfer large numbers of files to Azure Files, we recommend that you use AzCopy (for Windows; in preview for Linux and UNIX) or Azure PowerShell. These tools have been optimized for network transfer.

- **Why is my performance slow after I mount my Azure file share on Windows Server 2012 R2 or Windows 8.1?**

There is a known issue when mounting an Azure file share on Windows Server 2012 R2 and Windows 8.1. The issue was patched in the April 2014 cumulative update for Windows 8.1 and Windows Server 2012 R2. For optimum performance, ensure that all instances of Windows Server 2012 R2 and Windows 8.1 have this patch applied. (You should always receive Windows patches through Windows Update.) For more information, see the associated Microsoft Knowledge Base article [Slow performance when you access Azure Files from Windows 8.1 or Server 2012 R2](#).

Features and interoperability with other services

- **Can I use my Azure file share as a *File Share Witness* for my Windows Server Failover Cluster?**

Currently, this configuration is not supported for an Azure file share. For more information about how to set this up for Azure Blob storage, see [Deploy a Cloud Witness for a Failover Cluster](#).

- **Can I mount an Azure file share on an Azure Container instance?**

Yes, Azure file shares are a good option when you want to persist information beyond the lifetime of a container instance. For more information, see [Mount an Azure file share with Azure Container instances](#).

- **Is there a rename operation in the REST API?**

Not at this time.

- **Can I set up nested shares? In other words, a share under a share?**

No. The file share is the virtual driver that you can mount, so nested shares are not supported.

- **How do I use Azure Files with IBM MQ?**

IBM has released a document that helps IBM MQ customers configure Azure Files with the IBM service. For more information, see [How to set up an IBM MQ multi-instance queue manager with Microsoft Azure Files service](#).

See also

- [Troubleshoot Azure Files in Windows](#)
- [Troubleshoot Azure Files in Linux](#)
- [Troubleshoot Azure File Sync](#)

Create an Azure file share

2/25/2020 • 12 minutes to read • [Edit Online](#)

To create an Azure file share, you need to answer three questions about how you will use it:

- **What are the performance requirements for your Azure file share?**

Azure Files offers standard file shares, which are hosted on hard disk-based (HDD-based) hardware, and premium file shares, which are hosted on solid-state disk-based (SSD-based) hardware.

- **What size file share do you need?**

Standard file shares can span up to 100 TiB, however this feature is not enabled by default; if you need a file share that is larger than 5 TiB, you will need to enable the large file share feature for your storage account. Premium file shares can span up to 100 TiB without any special setting, however premium file shares are provisioned, rather than pay as you go like standard file shares. This means that provisioning a file share much larger than what you need will increase the total cost of storage.

- **What are your redundancy requirements for your Azure file share?**

Standard file shares offer locally-redundant (LRS), zone redundant (ZRS), geo-redundant (GRS), or geo-zone-redundant (GZRS) storage, however the large file share feature is only supported on locally redundant and zone redundant file shares. Premium file shares do not support any form of geo-redundancy.

Premium file shares are available with locally redundancy in most regions that offer storage accounts and with zone redundancy in a smaller subset of regions. To find out if premium file shares are currently available in your region, see the [products available by region](#) page for Azure. For information about regions that support ZRS, see [Azure Storage redundancy](#).

For more information on these three choices, see [Planning for an Azure Files deployment](#).

Prerequisites

- This article assumes that you have already created an Azure subscription. If you don't already have a subscription, then create a [free account](#) before you begin.
- If you intend to use Azure PowerShell, [install the latest version](#).
- If you intend to use the Azure CLI, [install the latest version](#).

Create a storage account

Azure file shares are deployed into *storage accounts*, which are top-level objects that represent a shared pool of storage. This pool of storage can be used to deploy multiple file shares.

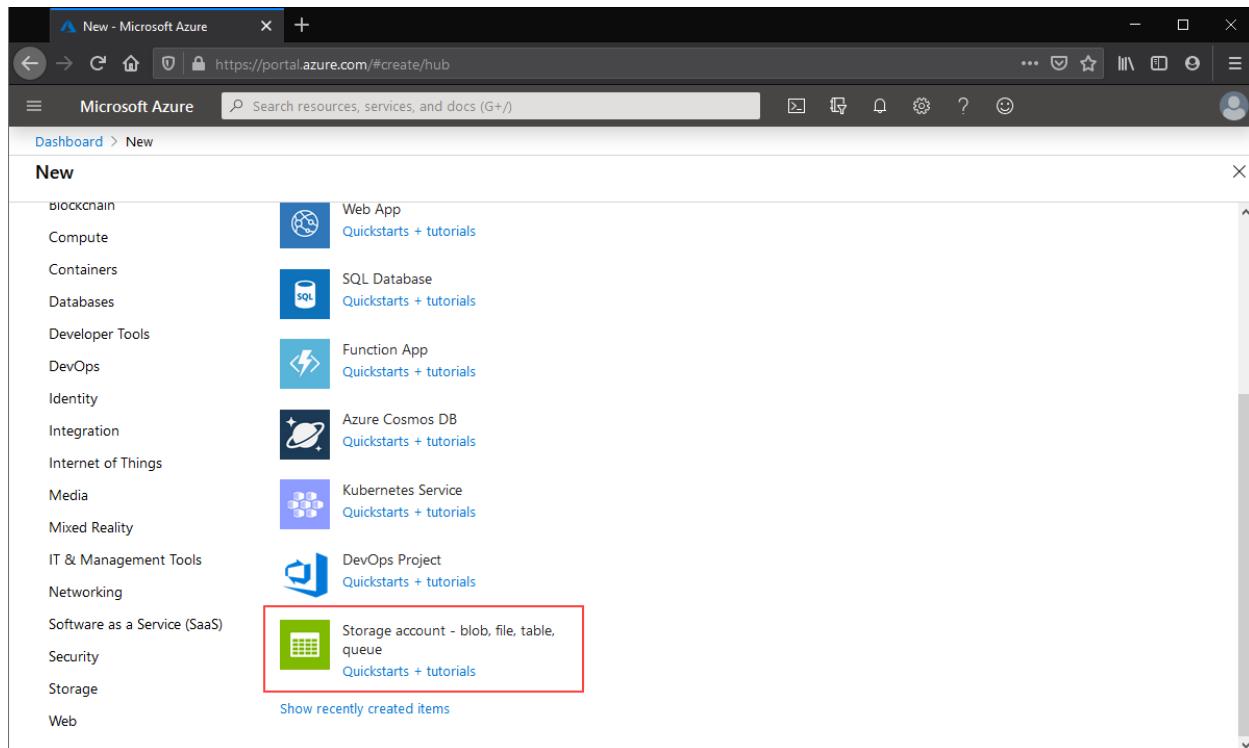
Azure supports multiple types of storage accounts for different storage scenarios customers may have, but there are two main types of storage accounts for Azure Files. Which storage account type you need to create depends on whether you want to create a standard file share or a premium file share:

- **General purpose version 2 (GPv2) storage accounts:** GPv2 storage accounts allow you to deploy Azure file shares on standard/hard disk-based (HDD-based) hardware. In addition to storing Azure file shares, GPv2 storage accounts can store other storage resources such as blob containers, queues, or tables.
- **FileStorage storage accounts:** FileStorage storage accounts allow you to deploy Azure file shares on premium/solid-state disk-based (SSD-based) hardware. FileStorage accounts can only be used to store

Azure file shares; no other storage resources (blob containers, queues, tables, etc.) can be deployed in a FileStorage account.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To create a storage account via the Azure portal, select **+ Create a resource** from the dashboard. In the resulting Azure Marketplace search window, search for **storage account** and select the resulting search result. This will lead to an overview page for storage accounts; select **Create** to proceed with the storage account creation wizard.



The Basics section

The first section to complete to create a storage account is labeled **Basics**. This contains all of the required fields to create a storage account. To create a GPv2 storage account, ensure the **Performance** radio button is set to *Standard* and the **Account kind** drop-down list is selected to *StorageV2 (general purpose v2)*.

Performance ⓘ Standard Premium

Account kind ⓘ

To create a FileStorage storage account, ensure the **Performance** radio button is set to *Premium* and the **Account kind** drop-down list is selected to *FileStorage*.

Performance ⓘ Standard Premium

Account kind ⓘ

ⓘ This account kind allows you to create Azure file shares with premium performance characteristics that can be tailored to your needs.
[Learn more about file storage ↗](#)

The other basics fields are independent from the choice of storage account:

- **Subscription:** The subscription for the storage account to be deployed into.
- **Resource group:** The resource group for the storage account to be deployed into. You may either create a

new resource group or use an existing 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.

- **Storage account name:** The name of the storage account resource to be created. This name must be globally unique, but otherwise can any name you desire. The storage account name will be used as the server name when you mount an Azure file share via SMB.
- **Location:** The region for the storage account to be deployed into. This can be the region associated with the resource group, or any other available region.
- **Replication:** Although this is labeled replication, this field actually means **redundancy**; this is the desired redundancy level: locally redundant (LRS), zone redundant (ZRS), geo-redundant (GRS), and geo-zone-redundancy. This drop-down list also contains read-access geo-redundancy (RA-GRS) and read-access geo-zone redundancy (RA-GZRS), which do not apply to Azure file shares; any file share created in a storage account with these selected will actually be either geo-redundant or geo-zone-redundant, respectively. Depending on your region or selected storage account type, some redundancy options may not be allowed.
- **Access tier:** This field does not apply to Azure Files, so you can choose either one of the radio buttons.

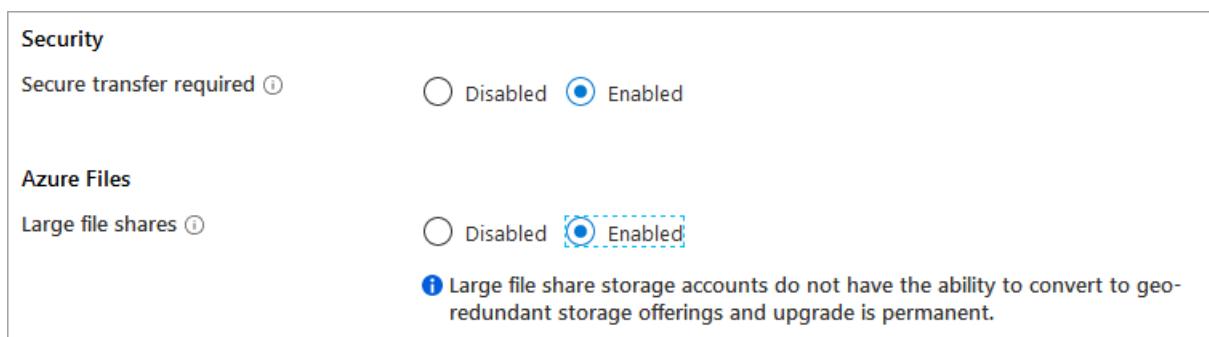
The Networking blade

The networking section allows you to configure networking options. These settings are optional for the creation of the storage account and can be configured later if desired. For more information on these options, see [Azure Files networking considerations](#).

The Advanced blade

The advanced section contains several important settings for Azure file shares:

- **Secure transfer required:** This field indicates whether the storage account requires encryption in transit for communication to the storage account. We recommend this is left enabled, however, if you require SMB 2.1 support, you must disable this. We recommend if you disable encryption that you constrain your storage account access to a virtual network with service endpoints and/or private endpoints.
- **Large file shares:** This field enables the storage account for file shares spanning up to 100 TiB. Enabling this feature will limit your storage account to only locally redundant and zone redundant storage options. Once a GPv2 storage account has been enabled for large file shares, you cannot disable the large file share capability. FileStorage storage accounts (storage accounts for premium file shares) do not have this option, as all premium file shares can scale up to 100 TiB.



The other settings that are available in the advanced tab (blob soft-delete, hierarchical namespace for Azure Data Lake storage gen 2, and NFSv3 for blob storage) do not apply to Azure Files.

Tags

Tags are name/value pairs that enable you to categorize resources and view consolidated billing by applying the same tag to multiple resources and resource groups. These are optional and can be applied after storage account creation.

Review + create

The final step to create the storage account is to select the **Create** button on the **Review + create** tab. This button won't be available if all of the required fields for a storage account are not filled.

Create file share

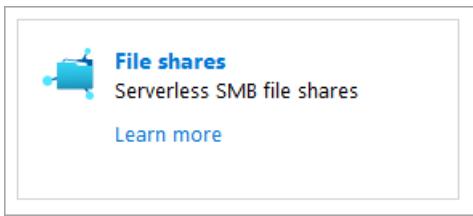
Once you've created your storage account, all that is left is to create your file share. This process is mostly the same regardless of whether you're using a premium file share or a standard file share. The primary difference is the **quota** and what it represents.

For standard file shares, it's an upper boundary of the Azure file share, beyond which end-users cannot go. The primary purpose for quota for a standard file share is budgetary: "I don't want this file share to grow beyond this point". If a quota is not specified, standard file share can span up to 100 TiB (or 5 TiB if the large file shares property is not set for a storage account).

For premium file shares, quota is overloaded to mean **provisioned size**. The provisioned size is the amount that you will be billed for, regardless of actual usage. When you provision a premium file share, you want to consider two factors: 1) the future growth of the share from a space utilization perspective and 2) the IOPS required for your workload. Every provisioned GiB entitles you to additional reserved and burst IOPS. For more information on how to plan for a premium file share, see [provisioning premium file shares](#).

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

If you just created your storage account, you can navigate to it from the deployment screen by selecting **Go to resource**. If you have previously created the storage account, you can navigate to it via the resource group containing it. Once in the storage account, select the tile labeled **File shares** (you can also navigate to **File shares** via the table of contents for the storage account).



In the file share listing, you should see any file shares you have previously created in this storage account; an empty table if no file shares have been created yet. Select **+ File share** to create a new file share.

The new file share blade should appear on the screen. Complete the fields in the new file share blade to create a file share:

- **Name:** the name of the file share to be created.
- **Quota:** The quota of the file share for standard file shares; the provisioned size of the file share for premium file shares.

Select **Create** to finish creating the new share. Note that if your storage account is in a virtual network, you will not be able to successfully create an Azure file share unless your client is also in the virtual network. You can also work around this point-in-time limitation by using the Azure PowerShell `New-AzRmStorageShare` cmdlet.

NOTE

The name of your file share must be all lowercase. For complete details about naming file shares and files, see [Naming and referencing shares, directories, files, and metadata](#).

Next steps

- [Plan for a deployment of Azure Files](#) or [Plan for a deployment of Azure File Sync](#).
- [Networking overview](#).

- Connect and mount a file share on [Windows](#), [macOS](#), and [Linux](#).

Enable and create large file shares

2/14/2020 • 4 minutes to read • [Edit Online](#)

When you enable large file shares on your storage account, your file shares can scale up to 100 TiB. You can enable this scaling on your existing storage accounts for your existing file shares.

Prerequisites

- If you don't have an Azure subscription, create a [free account](#) before you begin.
- If you intend to use the Azure CLI, [install the latest version](#).
- If you intend to use Azure PowerShell, [install the latest version](#).

Restrictions

For now, you can only use locally-redundant storage (LRS) or zone-redundant storage (ZRS) on large file share-enabled accounts. You can't use geo-zone-redundant storage (GZRS), geo-redundant storage (GRS), or read-access geo-redundant storage (RA-GRS). Enabling large file shares on an account is an irreversible process. After you enable it, you won't be able to convert your account to GZRS, GRS, or RA-GRS.

Create a new storage account

Portal

1. Sign in to the [Azure portal](#).
2. In the Azure portal, select **All services**.
3. In the list of resources, enter **Storage Accounts**. As you type, the list filters based on your input. Select **Storage Accounts**.
4. On the **Storage Accounts** window that appears, select **Add**.
5. Select the subscription that you'll use to create the storage account.
6. Under the **Resource group** field, select **Create new**. Enter a name for your new resource group.

7. Next, enter a name for your storage account. The name must be unique across Azure. The name also must be 3 to 24 characters in length, and it can only have numbers and lowercase letters.
8. Select a location for your storage account, and make sure it's [one of the supported replication for large file shares](#).
9. Set the replication to either **Locally redundant storage** or **Zone-redundant storage**.
10. Leave these fields at their default values:

FIELD	VALUE
Deployment model	Resource Manager
Performance	Standard
Account kind	StorageV2 (general-purpose v2)
Access tier	Hot

11. Select **Advanced**, and then select the **Enabled** option button to the right of **Large file shares**.
12. Select **Review + Create** to review your storage account settings and create the account.

Basics Networking **Advanced** Tags Review + create

Security

Secure transfer required ⓘ Disabled Enabled

Azure Files

Large file shares ⓘ Disabled Enabled

ⓘ Large file share storage accounts do not have the ability to convert to geo-redundant storage offerings and upgrade is permanent.

Data protection

Blob soft delete ⓘ Disabled Enabled

Data Lake Storage Gen2

Hierarchical namespace ⓘ Disabled Enabled

13. Select **Create**.

CLI

First, [install the latest version of the Azure CLI](#) so that you can enable large file shares.

To create a storage account with large file shares enabled, use the following command. Replace

`<yourStorageAccountName>`, `<yourResourceGroup>`, and `<yourDesiredRegion>` with your information.

```
## This command creates a large file share-enabled account. It will not support GZRS, GRS, or RA-GRS.
az storage account create --name <yourStorageAccountName> -g <yourResourceGroup> -l <yourDesiredRegion> --sku Standard_LRS --kind StorageV2 --enable-large-file-share
```

PowerShell

First, [install the latest version of PowerShell](#) so that you can enable large file shares.

To create a storage account with large file shares enabled, use the following command. Replace

`<yourStorageAccountName>`, `<yourResourceGroup>`, and `<yourDesiredRegion>` with your information.

```
## This command creates a large file share-enabled account. It will not support GZRS, GRS, or RA-GRS.
New-AzStorageAccount -ResourceGroupName <yourResourceGroup> -Name <yourStorageAccountName> -Location <yourDesiredRegion> -SkuName Standard_LRS -EnableLargeFileShare;
```

Enable large files shares on an existing account

You can also enable large file shares on your existing accounts. If you enable large file shares, you won't be able to convert to GZRS, GRS, or RA-GRS. Enabling large file shares is irreversible on this storage account.

Portal

1. Open the [Azure portal](#), and go to the storage account where you want to enable large file shares.
2. Open the storage account and select **Configuration**.
3. Select **Enabled** on **Large file shares**, and then select **Save**.
4. Select **Overview** and select **Refresh**.

The cost of your storage account depends on the usage and the options you choose below.
[Learn more](#)

Account kind
StorageV2 (general purpose v2)

Performance ⓘ
 Standard Premium

Secure transfer required * ⓘ
 Disabled Enabled

Access tier (default) ⓘ
 Cool Hot

Replication ⓘ
Locally-redundant storage (LRS)

Large file shares ⓘ
 Enabled Disabled

Identity-based Directory Service for Azure File Authentication ⓘ
None

Data Lake Storage Gen2
Hierarchical namespace ⓘ
 Disabled Enabled

You've now enabled large file shares on your storage account. Next, you must update existing share's quota to take advantage of increased capacity and scale.

If you receive the error message "Large file shares are not available for the account yet," your region might be in the middle of completing its rollout. Contact support if you have an urgent need for large file shares.

CLI

To enable large file shares on your existing account, use the following command. Replace `<yourStorageAccountName>` and `<yourResourceGroup>` with your information.

```
az storage account update --name <yourStorageAccountName> -g <yourResourceGroup> --enable-large-file-share
```

PowerShell

To enable large file shares on your existing account, use the following command. Replace `<yourStorageAccountName>` and `<yourResourceGroup>` with your information.

```
Set-AzStorageAccount -ResourceGroupName <yourResourceGroup> -Name <yourStorageAccountName> -  
EnableLargeFileShare
```

Create a large file share

After you've enabled large file shares on your storage account, you can create file shares in that account with higher quotas.

Portal

Creating a large file share is almost identical to creating a standard file share. The main difference is that you can

set a quota up to 100 TiB.

1. From your storage account, select **File shares**.
2. Select + **File share**.
3. Enter a name for your file share. You can also set the quota size you'd like, up to 100 TiB. Then select **Create**.

The screenshot shows the Azure Storage portal interface. On the left, there's a sidebar with various service links like Blob service, File service, and Container service. Under the 'File service' section, 'File shares' is selected and highlighted with a red box. The main pane shows a list of existing file shares under the heading 'Storage account: myexampleaccountname'. A search bar at the top says 'Search file shares by prefix'. On the right, a modal window titled 'File share' is open, prompting for a 'Name' (set to 'example-share') and a 'Quota' (set to '102400 GiB'). At the bottom of the modal are 'Create' and 'Discard' buttons, with 'Create' being highlighted in blue.

CLI

To create a large file share, use the following command. Replace `<yourStorageAccountName>`, `<yourStorageAccountKey>`, and `<yourFileShareName>` with your information.

```
az storage share create --account-name <yourStorageAccountName> --account-key <yourStorageAccountKey> --name <yourFileShareName>
```

PowerShell

To create a large file share, use the following command. Replace `<YourStorageAccountName>`, `<YourStorageAccountKey>`, and `<YourStorageAccountFileShareName>` with your information.

```
##Config
$storageAccountName = "<YourStorageAccountName>"
$storageAccountKey = "<YourStorageAccountKey>"
$shareName="<YourStorageAccountFileShareName>"
$cxt = New-AzStorageContext -StorageAccountName $storageAccountName -StorageAccountKey $storageAccountKey
New-AzStorageShare -Name $shareName -Context $cxt
```

Expand existing file shares

After you've enabled large file shares on your storage account, you can also expand existing file shares in that account to the higher quota.

Portal

1. From your storage account, select **File shares**.
2. Right-click your file share, and then select **Quota**.
3. Enter the new size that you want, and then select **OK**.

The screenshot shows the Azure Storage Explorer interface. At the top, there are buttons for 'File share' and 'Refresh'. Below that, it says 'Storage account: myexampleaccountname'. There is a search bar with the placeholder 'Search file shares by prefix'. The main area displays a table with columns: 'Name', 'Modified', and 'Quota'. A single row is shown for 'example-share', with its details: Modified on '10/15/2019, 8:33:44 AM' and Quota at '5 TiB'. To the right of the row is a three-dot menu icon. A context menu is open over the 'example-share' row, listing options: 'Properties' (selected), 'Connect', 'Quota' (highlighted with a red box), 'Edit metadata', 'Access policy', 'View snapshots', and 'Delete share'. Each option has a corresponding icon next to it.

CLI

To set the quota to the maximum size, use the following command. Replace <yourStorageAccountName>, <yourStorageAccountKey>, and <yourFileShareName> with your information.

```
az storage share update --account-name <yourStorageAccountName> --account-key <yourStorageAccountKey> --name <yourFileShareName> --quota 102400
```

PowerShell

To set the quota to the maximum size, use the following command. Replace <YourStorageAccountName>, <YourStorageAccountKey>, and <YourStorageAccountFileShareName> with your information.

```
##Config
$storageAccountName = "<YourStorageAccountName>"
$storageAccountKey = "<YourStorageAccountKey>"
$shareName="<YourStorageAccountFileShareName>"
$cxt = New-AzStorageContext -StorageAccountName $storageAccountName -StorageAccountKey $storageAccountKey
# update quota
Set-AzStorageShareQuota -ShareName $shareName -Context $cxt -Quota 102400
```

Next steps

- [Connect and mount a file share on Windows](#)
- [Connect and mount a file share on Linux](#)
- [Connect and mount a file share on macOS](#)

How to create an premium Azure file share

2/25/2020 • 5 minutes to read • [Edit Online](#)

Premium file shares are offered on solid-state disk (SSD) storage media and are useful for IO-intensive workloads, including hosting databases and high-performance computing (HPC). Premium file shares are hosted in a special purpose storage account kind, called a FileStorage account. Premium file shares are designed for high performance and enterprise scale applications, providing consistent low latency, high IOPS, and high throughput shares.

This article shows you how to create this new account type using [Azure portal](#), Azure PowerShell, and Azure CLI.

Prerequisites

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

Create a premium file share using the Azure portal

Sign in to Azure

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

Create a filestorage storage account

Now you're ready to create your 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.

1. In the Azure portal, select **Storage Accounts** on the left menu.

The screenshot shows the Microsoft Azure portal homepage. The left sidebar lists various services with 'Storage accounts' highlighted by a red box. The main content area features a grid of Azure services and a section titled 'Make the most out of Azure' with four cards: Microsoft Learn, Azure Monitor, Security Center, and Azure Advisor.

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. For **Performance** select **Premium**.
8. Select **Account kind** and choose **FileStorage**.
9. Leave **Replication** set to its default value of **Locally-redundant storage (LRS)**.

Azure Storage is a Microsoft-managed service providing cloud storage that is highly available, secure, durable, scalable, and redundant. Azure Storage includes Azure Blobs (objects), Azure Data Lake Storage Gen2, Azure Files, Azure Queues, and Azure Tables. The cost of your storage account depends on the usage and the options you choose below. [Learn more](#)

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

* Subscription

 * Resource group [Create new](#)

Instance details

The default deployment model is Resource Manager, which supports the latest Azure features. You may choose to deploy using the classic deployment model instead. [Choose classic deployment model](#)

* Storage account name ✓

* Location

Performance Standard Premium

Account kind !
 This account kind allows you to create Azure file shares with premium performance characteristics that can be tailored to your needs. [Learn more](#)

Replication !
 Accounts with the selected kind, replication, and performance type only support file shares. Blobs, tables, and queues will not be available.

[Review + create](#)[< Previous](#)[Next : Advanced >](#)

10. Select **Review + Create** to review your storage account settings and create the account.

11. Select **Create**.

Once your storage account resource has been created, navigate to it.

Create a premium file share

- In the left menu for the storage account, scroll to the **File service** section, then select **Files**.
- Select **File share** to create a premium file share.
- Enter a name and a desired quota for your file share, then select **Create**.

NOTE

Provisioned share sizes is specified by the share quota, file shares are billed on the provisioned size, refer to the [pricing page](#) for more details.

The screenshot shows the Azure Storage Account 'examplefilestorageacct' - Files blade. The 'File service' section is selected, and the 'Files' sub-section is highlighted. A modal window titled 'File share' is open, prompting to create a new file share. The 'Name' field is set to 'example-premium-file-share'. Below it, there's a note about provisioning: 'A premium file share is billed by provisioned share size, regardless of the used capacity.' It also specifies a minimum share size of 100 GiB and notes that provisioned share size is specified by share quota. The 'Quota' field is set to 100 GiB. At the bottom of the modal are 'Create' and 'Discard' buttons.

Clean up resources

If you would like to clean up the resources created in this article, you can simply delete the resource group. Deleting the resource group also deletes the associated storage account as well as any other resources associated with the resource group.

Create a premium file share using PowerShell

Create an account using PowerShell

First, install the latest version of the [PowerShellGet](#) module.

Then, upgrade your PowerShell module, sign in to your Azure subscription, create a resource group, and then create a storage account.

Upgrade your PowerShell module

To interact with a premium file share from with PowerShell, you'll need to install an Az.Storage module version 1.4.0, or the latest Az.Storage module.

Start by opening a PowerShell session with elevated permissions.

Install the Az.Storage module:

```
Install-Module Az.Storage -Repository PSGallery -AllowClobber -Force
```

Sign in to your Azure Subscription

Use the `Connect-AzAccount` command and follow the on-screen directions to authenticate.

```
Connect-AzAccount
```

Create a resource group

To create a new resource group with PowerShell, use the [New-AzResourceGroup](#) command:

```
# put resource group in a variable so you can use the same group name going forward,  
# without hardcoding it repeatedly  
$resourceGroup = "storage-how-to-resource-group"  
$location = "westus2"  
New-AzResourceGroup -Name $resourceGroup -Location $location
```

Create a FileStorage storage account

To create a FileStorage storage account from PowerShell, use the [New-AzStorageAccount](#) command:

```
$storageAcct = New-AzStorageAccount -ResourceGroupName $resourceGroup -Name "fileshowto" -SkuName  
"Premium_LRS" -Location "westus2" -Kind "FileStorage"
```

Create a premium file share

Now that you have a FileStorage account, you can create a premium file share. Use the [New-AzStorageShare](#) cmdlet to create one.

NOTE

Provisioned share sizes is specified by the share quota, file shares are billed on the provisioned size, refer to the [pricing page](#) for more details.

```
New-AzStorageShare `  
-Name myshare `  
-Context $storageAcct.Context
```

Clean up resources

To remove the resource group and its associated resources, including the new storage account, use the [Remove-AzResourceGroup](#) command:

```
Remove-AzResourceGroup -Name $resourceGroup
```

Create a premium file share using Azure CLI

To start Azure Cloud Shell, sign in to the [Azure portal](#).

If you want to log into your local installation of the CLI, first make sure you have the latest version, then run the login command:

```
az login
```

Create a resource group

To create a new resource group with Azure CLI, use the [az group create](#) command.

```
az group create `  
--name files-howto-resource-group `  
--location westus2
```

Create a FileStorage storage account

To create a FileStorage storage account from the Azure CLI, use the [az storage account create](#) command.

```
az storage account create `  
  --name fileshowto `  
  --resource-group files-howto-resource-group `  
  --location westus `  
  --sku Premium_LRS `  
  --kind FileStorage
```

Get the storage account key

Storage account keys control access to resources in a storage account, in this article, we use the key in order to create a premium file share. The keys are automatically created when you create a storage account. You can get the storage account keys for your storage account by using the [az storage account keys list](#) command:

```
STORAGEKEY=$(az storage account keys list \  
  --resource-group "myResourceGroup" \  
  --account-name $STORAGEACCT \  
  --query "[0].value" | tr -d '')
```

Create a premium file share

Now that you have a filestorage account, you can create a premium file share. Use the [az storage share create](#) command to create one.

NOTE

Provisioned share sizes is specified by the share quota, file shares are billed on the provisioned size, refer to the [pricing page](#) for more details.

```
az storage share create \  
  --account-name $STORAGEACCT \  
  --account-key $STORAGEKEY \  
  --name "myshare"
```

Clean up resources

To remove the resource group and its associated resources, including the new storage account, use the [az group delete](#) command.

```
az group delete --name myResourceGroup
```

Next steps

In this article, you've created a premium file share. To learn about the performance this account offers, continue to the performance tier section of the planning guide.

[File share tiers](#)

How to deploy Azure Files

2/25/2020 • 6 minutes to read • [Edit Online](#)

Azure Files offers fully managed file shares in the cloud that are accessible via the industry standard SMB protocol. This article will show you how to practically deploy Azure Files within your organization.

We strongly recommend reading [Planning for an Azure Files deployment](#) prior to following the steps in this article.

Prerequisites

This article assumes that you have already completed the following steps:

- Created an Azure Storage Account with your desired resiliency and encryption options, in the region you desire. See [Create a Storage Account](#) for step-by-step directions on how to create a Storage Account.
- Created an Azure file share with your desired quota in your Storage Account. See [Create a file share](#) for step-by-step directions on how to create a file share.

Transfer data into Azure Files

You may wish to migrate existing file shares, such as those stored on-premises, to your new Azure file share. This section will show you how to move data into an Azure file share via several popular methods detailed from the [planning guide](#)

Azure File Sync

Azure File Sync allows you to centralize your organization's file shares in Azure Files without giving up the flexibility, performance, and compatibility of an on-premises file server. It does this by transforming your Windows Servers into a quick cache of your Azure file share. You can use any protocol available on Windows Server to access your data locally (including SMB, NFS, and FTPS) and you can have as many caches as you need across the world.

Azure File Sync may be used to migrate data into an Azure file share, even if the sync mechanism isn't desired for long-term use. More information on how to use Azure File Sync to transfer data into Azure file share can be found in [Planning for an Azure File Sync deployment](#) and [How to deploy Azure File Sync](#).

Azure Import/Export

The Azure Import/Export service allows you to securely transfer large amounts of data into an Azure file share by shipping hard disk drives to an Azure datacenter. See [Use the Microsoft Azure Import/Export service to transfer data to Azure storage](#) for a more detailed overview of the service.

NOTE

The Azure Import/Export service does not support the export of files from an Azure file share at this time.

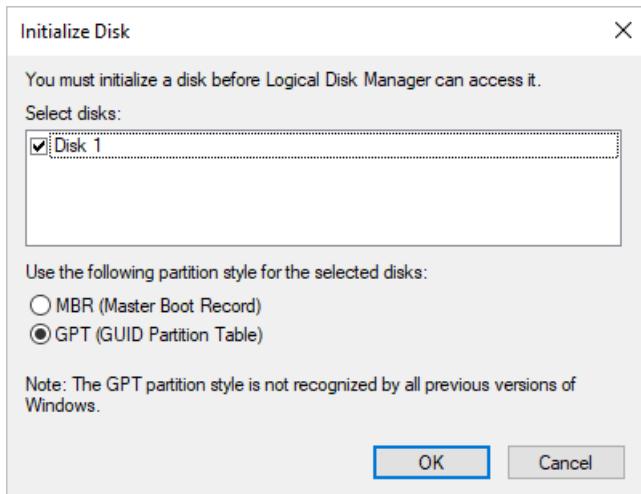
The following steps will import data from an on-premises location to your Azure file share.

1. Procure the required number of hard disks to mail into Azure. Hard disks may be of any disk size, but must be either a 2.5" or 3.5" SSD or HDD supporting the SATA II or SATA III standard.
2. Connect and mount each disk on the server/PC doing the data transfer. For optimal performance, we recommend running the on-premises export job locally on the server that contains the data. In some cases,

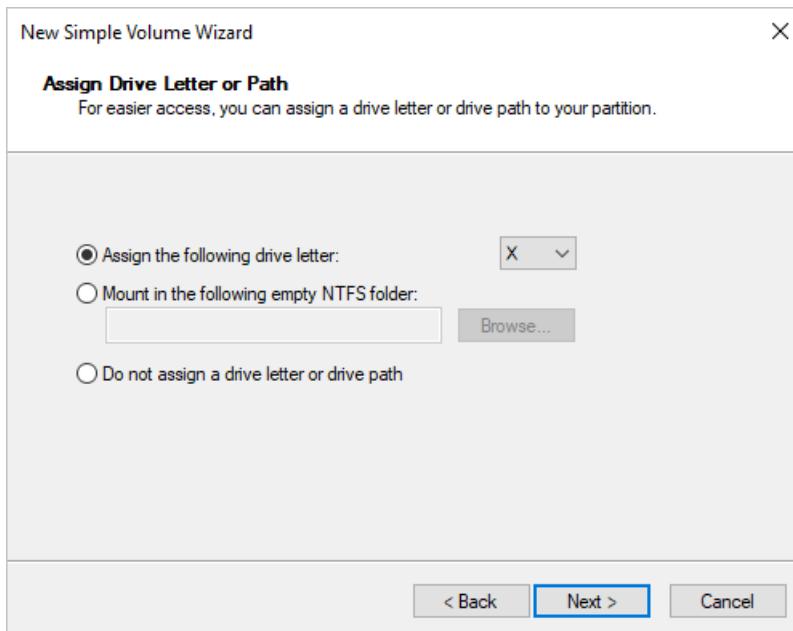
such as when the file server that serves the data is a NAS device, this may not be possible. In that case, it is perfectly acceptable to mount each disk on a PC.

3. Ensure each drive is online, initialized, and is assigned a drive letter. To do bring a drive online, initialize, and assign a drive letter, open the Disk Management MMC snap-in (diskmgmt.msc).

- To bring a disk online (if it's not already online), right-click on the disk in the lower pane of the Disk Management MMC and select "Online".
- To initialize a disk, right-click on the disk in the lower pane (after the disk is online), and select "Initialize". Be sure to select "GPT" when asked.



- To assign a drive letter to the disk, right-click on the "unallocated" space of the online and initialized disk, and click "New Simple Volume". This will allow you to assign drive letter. Note that you do not need to format the volume as this will be done later.



4. Create the dataset CSV file. The dataset CSV file is a mapping between the path to the data on-premises and the desired Azure file share the data should be copied to. For example, the following dataset CSV file maps an on-premises file share ("F:\shares\scratch") to an Azure file share ("MyAzureFileShare"):

```
BasePath,DstItemPathOrPrefix,ItemType,Disposition,MetadataFile,PropertiesFile  
"F:\shares\scratch\", "MyAzureFileShare/", file, rename, "None", None
```

Multiple shares with a Storage Account may be specified. See [Prepare the dataset CSV file](#) for more

information.

5. Create the driveset CSV file. The driveset CSV file lists the disks available to the on-premises export agent. For example, the following driveset CSV file lists `x:`, `y:`, and `z:` drives to be used in the on-premises export job:

```
DriveLetter,FormatOption,SilentOrPromptOnFormat,Encryption,ExistingBitLockerKey  
X,Format,SilentMode,Encrypt,  
Y,Format,SilentMode,Encrypt,  
Z,Format,SilentMode,Encrypt,
```

See [Prepare the driveset CSV file](#) for more information.

6. Use the [WAImpoerExport Tool](#) to copy your data to one or more hard drives.

```
WAImpoerExport.exe PrepImport /j:<JournalFile> /id:<SessionId> [/logdir:<LogDirectory>] [/sk:<StorageAccountKey>] [/silentmode] [/InitialDriveSet:<driveset.csv>] DataSet:<dataset.csv>
```

WARNING

Do not modify the data on the hard disk drives or the journal file after completing disk preparation.

7. [Create an import job](#).

Robocopy

Robocopy is a well known copy tool that ships with Windows and Windows Server. Robocopy may be used to transfer data into Azure Files by mounting the file share locally, and then using the mounted location as the destination in the Robocopy command. Using Robocopy is quite simple:

1. [Mount your Azure file share](#). For optimal performance, we recommend mounting the Azure file share locally on the server that contains the data. In some cases, such as when the file server that serves the data is a NAS device, this may not be possible. In that case, it is perfectly acceptable to mount the Azure file share on a PC. In this example, `net use` is used at the command line to mount the file share:

```
net use <desired-drive-letter>: \\<storage-account-name>.file.core.windows.net\<share-name> <storage-account-key> /user:Azure\<storage-account-name>
```

2. Use `robocopy` at the command line to move data to the Azure file share:

```
robocopy <path-to-local-share> <path-to-azure-file-share> /E /Z /MT:32
```

Robocopy has a significant number of options to modify the copy behavior as desired. For more information, view the [Robocopy](#) manual page.

AzCopy

AzCopy is a command-line utility designed for copying data to and from Azure Files, as well as Azure Blob storage, using simple commands with optimal performance. Using AzCopy is easy:

1. Download the [latest version of AzCopy on Windows](#) or [Linux](#).
2. Use `azcopy` at the command line to move data to the Azure file share. The syntax on Windows is as follows:

```
azcopy /Source:<path-to-local-share> /Dest:https://<storage-account>.file.core.windows.net/<file-share>/ /DestKey:<storage-account-key> /S
```

On Linux, the command syntax is a little different:

```
azcopy --source <path-to-local-share> --destination https://<storage-account>.file.core.windows.net/<file-share>/ --dest-key <storage-account-key> --recursive
```

AzCopy has a significant number of options to modify the copy behavior as desired. For more information, view [AzCopy on Windows](#) and [AzCopy on Linux](#).

Automatically mount on needed PCs/Servers

To replace an on-premises file share, it is helpful to pre-mount the shares on the machines it will be used on. This can be done automatically on a list of machines.

NOTE

Mounting an Azure file share requires using the storage account key as the password, therefore we only recommend mounting in trusted environments.

Windows

PowerShell can be used run the mount command on multiple PCs. In the following example, `$computers` is manually populated, but you can generate the list of computers to mount automatically. For example, you can populate this variable with results from Active Directory.

```
$computer = "MyComputer1", "MyComputer2", "MyComputer3", "MyComputer4"
$computer | ForEach-Object { Invoke-Command -ComputerName $_ -ScriptBlock { net use <desired-drive-letter>: \\<storage-account-name>.file.core.windows.net\<share-name> <storage-account-key> /user:Azure\<storage-account-name> /PERSISTENT:YES } }
```

Linux

A simple bash script combined with SSH can yield the same result in the following example. The `$computer` variable is similarly left to be populated by the user:

```
computer = ("MyComputer1" "MyComputer2" "MyComputer3" "MyComputer4")
for item in "${computer[@]}"
do
    ssh $item "sudo bash -c 'echo \"//<storage-account-name>.file.core.windows.net/<share-name> /mymountpoint
cifs vers=3.0,username=<storage-account-name>,password=<storage-account-key>,dir_mode=0777,file_mode=0777,serverino\" >> /etc/fstab'", "sudo mount -a"
done
```

Next steps

- [Plan for an Azure File Sync deployment](#)
- [Troubleshoot Azure Files on Windows](#)
- [Troubleshoot Azure Files on Linux](#)

Deploy Azure File Sync

2/25/2020 • 21 minutes to read • [Edit Online](#)

Use Azure File Sync to centralize your organization's file shares in Azure Files, while keeping the flexibility, performance, and compatibility of an on-premises file server. Azure File Sync transforms Windows Server into a quick cache of your Azure file share. You can use any protocol that's available on Windows Server to access your data locally, including SMB, NFS, and FTPS. You can have as many caches as you need across the world.

We strongly recommend that you read [Planning for an Azure Files deployment](#) and [Planning for an Azure File Sync deployment](#) before you complete the steps described in this article.

Prerequisites

- An Azure file share in the same region that you want to deploy Azure File Sync. For more information, see:
 - [Region availability for Azure File Sync](#).
 - [Create a file share](#) for a step-by-step description of how to create a file share.
- At least one supported instance of Windows Server or Windows Server cluster to sync with Azure File Sync. For more information about supported versions of Windows Server, see [Interoperability with Windows Server](#).
- The Az PowerShell module may be used with either PowerShell 5.1 or PowerShell 6+. You may use the Az PowerShell module for Azure File Sync on any supported system, including non-Windows systems, however the server registration cmdlet must always be run on the Windows Server instance you are registering (this can be done directly or via PowerShell remoting). On Windows Server 2012 R2, you can verify that you are running at least PowerShell 5.1.* by looking at the value of the **PSVersion** property of the **\$PSVersionTable** object:

```
$PSVersionTable.PSVersion
```

If your PSVersion value is less than 5.1.*^{*}, as will be the case with most fresh installations of Windows Server 2012 R2, you can easily upgrade by downloading and installing [Windows Management Framework \(WMF\) 5.1](#). The appropriate package to download and install for Windows Server 2012 R2 is [Win8.1AndW2K12R2-KB*****-x64.msu](#).

PowerShell 6+ can be used with any supported system, and can be downloaded via its [GitHub page](#).

IMPORTANT

If you plan to use the Server Registration UI, rather than registering directly from PowerShell, you must use PowerShell 5.1.

- If you have opted to use PowerShell 5.1, ensure that at least .NET 4.7.2 is installed. Learn more about [.NET Framework versions and dependencies](#) on your system.

IMPORTANT

If you are installing .NET 4.7.2+ on Windows Server Core, you must install with the `quiet` and `norestart` flags or the installation will fail. For example, if installing .NET 4.8, the command would look like the following:

```
Start-Process -FilePath "ndp48-x86-x64-allos-enu.exe" -ArgumentList "/q /norestart" -Wait
```

- The Az PowerShell module, which can be installed by following the instructions here: [Install and configure Azure PowerShell](#).

NOTE

The Az.StorageSync module is now installed automatically when you install the Az PowerShell module.

Prepare Windows Server to use with Azure File Sync

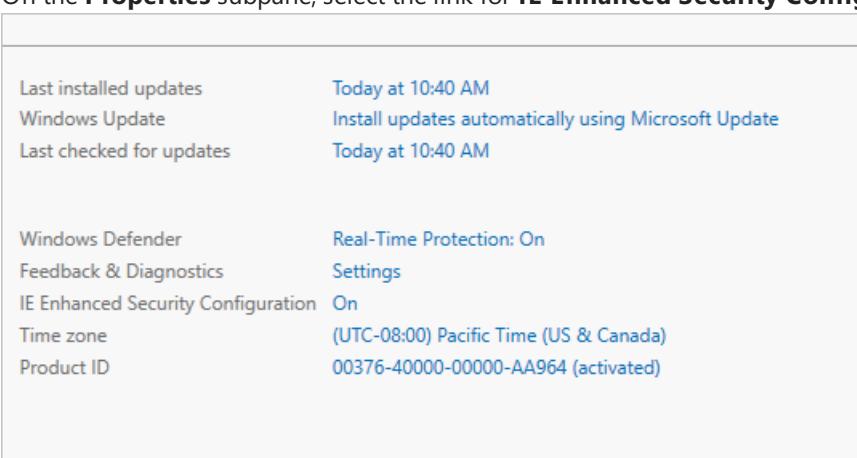
For each server that you intend to use with Azure File Sync, including each server node in a Failover Cluster, disable **Internet Explorer Enhanced Security Configuration**. This is required only for initial server registration. You can re-enable it after the server has been registered.

- [Portal](#)
- [PowerShell](#)

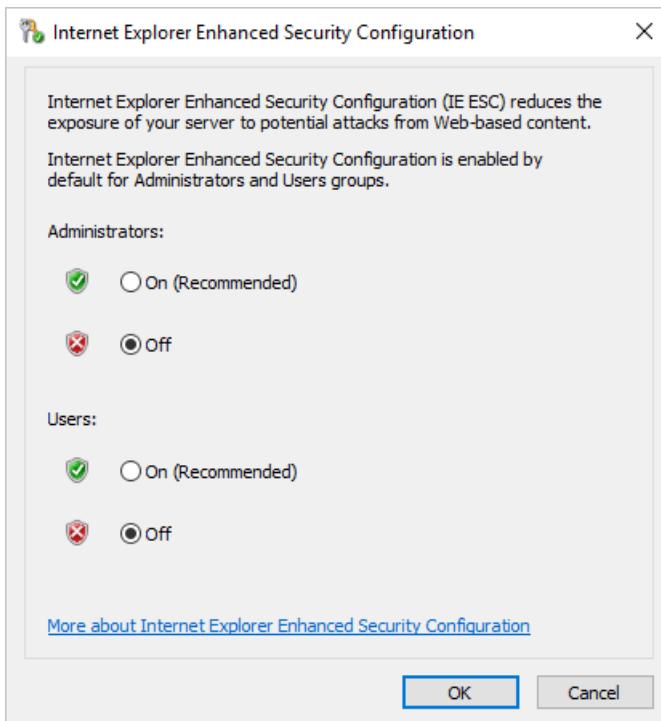
NOTE

You can skip this step if you're deploying Azure File Sync on Windows Server Core.

1. Open Server Manager.
2. Click **Local Server**:
A screenshot of the Windows Server Manager navigation pane. It shows three items: 'Dashboard' (grey icon), 'Local Server' (blue icon with a white outline, indicating it is selected), and 'All Servers' (grey icon).
3. On the **Properties** subpane, select the link for **IE Enhanced Security Configuration**.



4. In the **Internet Explorer Enhanced Security Configuration** dialog box, select **Off** for **Administrators** and **Users**:



Deploy the Storage Sync Service

The deployment of Azure File Sync starts with placing a **Storage Sync Service** resource into a resource group of your selected subscription. We recommend provisioning as few of these as needed. You will create a trust relationship between your servers and this resource and a server can only be registered to one Storage Sync Service. As a result, it is recommended to deploy as many storage sync services as you need to separate groups of servers. Keep in mind that servers from different storage sync services cannot sync with each other.

NOTE

The Storage Sync Service inherits access permissions from the subscription and resource group it has been deployed into. We recommend that you carefully check who has access to it. Entities with write access can start syncing new sets of files from servers registered to this storage sync service and cause data to flow to Azure storage that is accessible to them.

- [Portal](#)
- [PowerShell](#)

To deploy a Storage Sync Service, go to the [Azure portal](#), click *Create a resource* and then search for Azure File Sync. In the search results, select **Azure File Sync**, and then select **Create** to open the **Deploy Storage Sync** tab.

On the pane that opens, enter the following information:

- **Name:** A unique name (per subscription) for the Storage Sync Service.
- **Subscription:** The subscription in which you want to create the Storage Sync Service. Depending on your organization's configuration strategy, you might have access to one or more subscriptions. An Azure subscription is the most basic container for billing for each cloud service (such as Azure Files).
- **Resource group:** A resource group is a logical group of Azure resources, such as a storage account or a Storage Sync Service. You can create a new resource group or use an existing resource group for Azure File Sync. (We recommend using resource groups as containers to isolate resources logically for your organization, such as grouping HR resources or resources for a specific project.)
- **Location:** The region in which you want to deploy Azure File Sync. Only supported regions are available in this list.

When you are finished, select **Create** to deploy the Storage Sync Service.

Install the Azure File Sync agent

The Azure File Sync agent is a downloadable package that enables Windows Server to be synced with an Azure file share.

- [Portal](#)
- [PowerShell](#)

You can download the agent from the [Microsoft Download Center](#). When the download is finished, double-click the MSI package to start the Azure File Sync agent installation.

IMPORTANT

If you intend to use Azure File Sync with a Failover Cluster, the Azure File Sync agent must be installed on every node in the cluster. Each node in the cluster must be registered to work with Azure File Sync.

We recommend that you do the following:

- Leave the default installation path (C:\Program Files\Azure\StorageSyncAgent), to simplify troubleshooting and server maintenance.
- Enable Microsoft Update to keep Azure File Sync up to date. All updates, to the Azure File Sync agent, including feature updates and hotfixes, occur from Microsoft Update. We recommend installing the latest update to Azure File Sync. For more information, see [Azure File Sync update policy](#).

When the Azure File Sync agent installation is finished, the Server Registration UI automatically opens. You must have a Storage Sync Service before registering; see the next section on how to create a Storage Sync Service.

Register Windows Server with Storage Sync Service

Registering your Windows Server with a Storage Sync Service establishes a trust relationship between your server (or cluster) and the Storage Sync Service. A server can only be registered to one Storage Sync Service and can sync with other servers and Azure file shares associated with the same Storage Sync Service.

NOTE

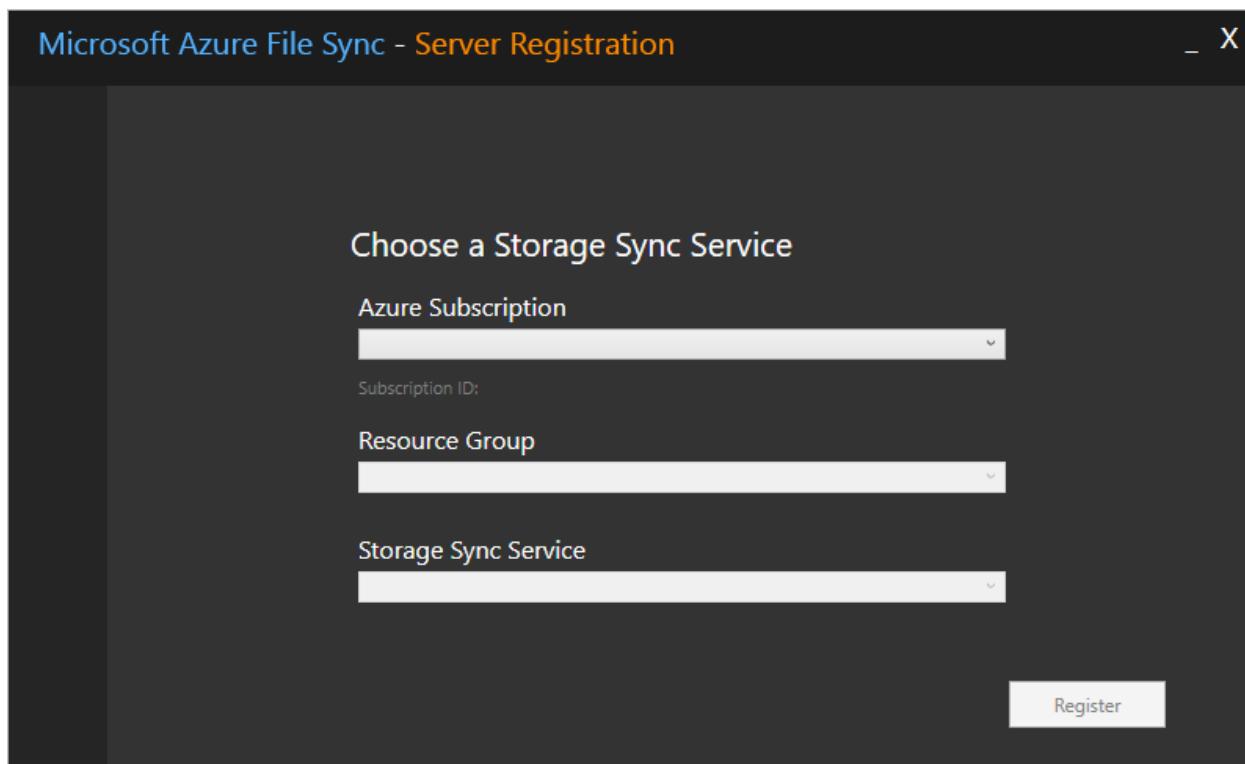
Server registration uses your Azure credentials to create a trust relationship between the Storage Sync Service and your Windows Server, however subsequently the server creates and uses its own identity that is valid as long as the server stays registered and the current Shared Access Signature token (Storage SAS) is valid. A new SAS token cannot be issued to the server once the server is unregistered, thus removing the server's ability to access your Azure file shares, stopping any sync.

- [Portal](#)
- [PowerShell](#)

The Server Registration UI should open automatically after installation of the Azure File Sync agent. If it doesn't, you can open it manually from its file location: C:\Program

Files\Azure\StorageSyncAgent\ServerRegistration.exe. When the Server Registration UI opens, select **Sign-in** to begin.

After you sign in, you are prompted for the following information:



- **Azure Subscription:** The subscription that contains the Storage Sync Service (see [Deploy the Storage Sync Service](#)).
- **Resource Group:** The resource group that contains the Storage Sync Service.
- **Storage Sync Service:** The name of the Storage Sync Service with which you want to register.

After you have selected the appropriate information, select **Register** to complete the server registration. As part of the registration process, you are prompted for an additional sign-in.

Create a sync group and a cloud endpoint

A sync group defines the sync topology for a set of files. Endpoints within a sync group are kept in sync with each other. A sync group must contain one cloud endpoint, which represents an Azure file share and one or more server endpoints. A server endpoint represents a path on a registered server. A server can have server endpoints in multiple sync groups. You can create as many sync groups as you need to appropriately describe your desired sync topology.

A cloud endpoint is a pointer to an Azure file share. All server endpoints will sync with a cloud endpoint, making the cloud endpoint the hub. The storage account for the Azure file share must be located in the same region as the Storage Sync Service. The entirety of the Azure file share will be synced, with one exception: A special folder, comparable to the hidden "System Volume Information" folder on an NTFS volume, will be provisioned. This directory is called ".SystemShareInformation". It contains important sync metadata that will not sync to other endpoints. Do not use or delete it!

IMPORTANT

You can make changes to any cloud endpoint or server endpoint in the sync group and have your files synced to the other endpoints in the sync group. If you make a change to the cloud endpoint (Azure file share) directly, changes first need to be discovered by an Azure File Sync change detection job. A change detection job is initiated for a cloud endpoint only once every 24 hours. For more information, see [Azure Files frequently asked questions](#).

- [Portal](#)
- [PowerShell](#)

To create a sync group, in the [Azure portal](#), go to your Storage Sync Service, and then select **+ Sync group**:

In the pane that opens, enter the following information to create a sync group with a cloud endpoint:

- **Sync group name:** The name of the sync group to be created. This name must be unique within the Storage Sync Service, but can be any name that is logical for you.
- **Subscription:** The subscription where you deployed the Storage Sync Service in [Deploy the Storage Sync Service](#).
- **Storage account:** If you select **Select storage account**, another pane appears in which you can select the storage account that has the Azure file share that you want to sync with.
- **Azure file share:** The name of the Azure file share with which you want to sync.

Create a server endpoint

A server endpoint represents a specific location on a registered server, such as a folder on a server volume. A server endpoint must be a path on a registered server (rather than a mounted share), and to use cloud tiering, the path must be on a non-system volume. Network attached storage (NAS) is not supported.

- [Portal](#)
- [PowerShell](#)

To add a server endpoint, go to the newly created sync group and then select **Add server endpoint**.

In the **Add server endpoint** pane, enter the following information to create a server endpoint:

- **Registered server:** The name of the server or cluster where you want to create the server endpoint.
- **Path:** The Windows Server path to be synced as part of the sync group.
- **Cloud Tiering:** A switch to enable or disable cloud tiering. With cloud tiering, infrequently used or accessed files can be tiered to Azure Files.
- **Volume Free Space:** The amount of free space to reserve on the volume on which the server endpoint is located. For example, if volume free space is set to 50% on a volume that has a single server endpoint, roughly half the amount of data is tiered to Azure Files. Regardless of whether cloud tiering is enabled, your Azure file share always has a complete copy of the data in the sync group.

To add the server endpoint, select **Create**. Your files are now kept in sync across your Azure file share and

Windows Server.

Configure firewall and virtual network settings

Portal

If you'd like to configure your Azure File sync to work with firewall and virtual network settings, do the following:

1. From the Azure portal, navigate to the storage account you want to secure.
2. Select the **Firewalls and virtual networks** button on the left menu.
3. Select **Selected networks** under **Allow access from**.
4. Make sure your servers IP or virtual network is listed under the appropriate section.
5. Make sure **Allow trusted Microsoft services to access this storage account** is checked.
6. Select **Save** to save your settings.

The screenshot shows the Azure portal interface for managing firewalls and virtual networks. On the left, there's a navigation sidebar with various storage account settings like Access keys, Geo-replication, and Shared access signature. The 'Firewalls and virtual networks' option is highlighted with a red box. The main content area has a header 'renashstorageacct - Firewalls and virtual networks'. It includes a 'Save' button, a 'Discard' button, and a 'Refresh' button. A section titled 'Allow access from' has a radio button for 'All networks' and another for 'Selected networks', which is selected and highlighted with a red box. Below this is a note about configuring network security for storage accounts. The 'Virtual networks' section contains a table with columns 'VIRTUAL NETWORK', 'SUBNET', 'ADDRESS RANGE', 'ENDPOINT STATUS', 'RESOURCE GROUP', and 'SUBSCRIPTION'. A note says 'No network selected.' The 'Firewall' section has a table for 'ADDRESS RANGE' with two entries: '131.107.147.209' and '40.70.23.209'. The 'Exceptions' section contains a checkbox for 'Allow trusted Microsoft services to access this storage account' which is checked and highlighted with a red box. There are also other unchecked options for allowing read access to storage logging and metrics.

Onboarding with Azure File Sync

The recommended steps to onboard on Azure File Sync for the first with zero downtime while preserving full file fidelity and access control list (ACL) are as follows:

1. Deploy a Storage Sync Service.
2. Create a sync group.
3. Install Azure File Sync agent on the server with the full data set.
4. Register that server and create a server endpoint on the share.
5. Let sync do the full upload to the Azure file share (cloud endpoint).
6. After the initial upload is complete, install Azure File Sync agent on each of the remaining servers.
7. Create new file shares on each of the remaining servers.
8. Create server endpoints on new file shares with cloud tiering policy, if desired. (This step requires additional storage to be available for the initial setup.)
9. Let Azure File Sync agent do a rapid restore of the full namespace without the actual data transfer. After the full namespace sync, sync engine will fill the local disk space based on the cloud tiering policy for the server endpoint.
10. Ensure sync completes and test your topology as desired.
11. Redirect users and applications to this new share.

12. You can optionally delete any duplicate shares on the servers.

If you don't have extra storage for initial onboarding and would like to attach to the existing shares, you can pre-seed the data in the Azure files shares. This approach is suggested, if and only if you can accept downtime and absolutely guarantee no data changes on the server shares during the initial onboarding process.

1. Ensure that data on any of the servers can't change during the onboarding process.
2. Pre-seed Azure file shares with the server data using any data transfer tool over the SMB for example, Robocopy, direct SMB copy. Since AzCopy does not upload data over the SMB so it can't be used for pre-seeding.
3. Create Azure File Sync topology with the desired server endpoints pointing to the existing shares.
4. Let sync finish reconciliation process on all endpoints.
5. Once reconciliation is complete, you can open shares for changes.

Currently, pre-seeding approach has a few limitations -

- Full fidelity on files is not preserved. For example, files lose ACLs and timestamps.
- Data changes on the server before sync topology is fully up and running can cause conflicts on the server endpoints.
- After the cloud endpoint is created, Azure File Sync runs a process to detect the files in the cloud before starting the initial sync. The time taken to complete this process varies depending on the various factors like network speed, available bandwidth, and number of files and folders. For the rough estimation in the preview release, detection process runs approximately at 10 files/sec. Hence, even if pre-seeding runs fast, the overall time to get a fully running system may be significantly longer when data is pre-seeded in the cloud.

Self-service restore through Previous Versions and VSS (Volume Shadow Copy Service)

IMPORTANT

The following information can only be used with version 9 (or above) of the storage sync agent. Versions lower than 9 will not have the StorageSyncSelfService cmdlets.

Previous Versions is a Windows feature that allows you to utilize server-side VSS snapshots of a volume to present restorable versions of a file to an SMB client. This enables a powerful scenario, commonly referred to as self-service restore, directly for information workers instead of depending on the restore from an IT admin.

VSS snapshots and Previous Versions work independently of Azure File Sync. However, cloud tiering must be set to a compatible mode. Many Azure File Sync server endpoints can exist on the same volume. You have to make the following PowerShell call per volume that has even one server endpoint where you plan to or are using cloud tiering.

```
Import-Module '<SyncAgentInstallPath>\StorageSync.Management.ServerCmdlets.dll'  
Enable-StorageSyncSelfServiceRestore [-DriveLetter] <string> [[-Force]]
```

VSS snapshots are taken of an entire volume. By default, up to 64 snapshots can exist for a given volume, granted there is enough space to store the snapshots. VSS handles this automatically. The default snapshot schedule takes two snapshots per day, Monday through Friday. That schedule is configurable via a Windows Scheduled Task. The above PowerShell cmdlet does two things:

1. It configures Azure File Sync's cloud tiering on the specified volume to be compatible with previous versions and guarantees that a file can be restored from a previous version, even if it was tiered to the cloud on the server.

2. It enables the default VSS schedule. You can then decide to modify it later.

NOTE

There are two important things to note:

- If you use the -Force parameter, and VSS is currently enabled, then it will overwrite the current VSS snapshot schedule and replace it with the default schedule. Ensure you save your custom configuration before running the cmdlet.
- If you are using this cmdlet on a cluster node, you must also run it on all the other nodes in the cluster!

In order to see if self-service restore compatibility is enabled, you can run the following cmdlet.

```
Get-StorageSyncSelfServiceRestore [[-Driveletter] <string>]
```

It will list all volumes on the server as well as the number of cloud tiering compatible days for each. This number is automatically calculated based on the maximum possible snapshots per volume and the default snapshot schedule. So by default, all previous versions presented to an information worker can be used to restore from. The same is true if you change the default schedule to take more snapshots. However, if you change the schedule in a way that will result in an available snapshot on the volume that is older than the compatible days value, then users will not be able to use this older snapshot (previous version) to restore from.

NOTE

Enabling self-service restore can have an impact on your Azure storage consumption and bill. This impact is limited to files currently tiered on the server. Enabling this feature ensures that there is a file version available in the cloud that can be referenced via a previous versions (VSS snapshot) entry.

If you disable the feature, the Azure storage consumption will slowly decline until the compatible days window has passed. There is no way to speed this up.

The default maximum number of VSS snapshots per volume (64) as well as the default schedule to take them, result in a maximum of 45 days of previous versions an information worker can restore from, depending on how many VSS snapshots you can store on your volume.

If max. 64 VSS snapshots per volume is not the correct setting for you, you can [change that value via a registry key](#). For the new limit to take effect, you need to re-run the cmdlet to enable previous version compatibility on every volume it was previously enabled, with the -Force flag to take the new maximum number of VSS snapshots per volume into account. This will result in a newly calculated number of compatible days. Please note that this change will only take effect on newly tiered files and overwrite any customizations on the VSS schedule you might have made.

Migrate a DFS Replication (DFS-R) deployment to Azure File Sync

To migrate a DFS-R deployment to Azure File Sync:

1. Create a sync group to represent the DFS-R topology you are replacing.
2. Start on the server that has the full set of data in your DFS-R topology to migrate. Install Azure File Sync on that server.
3. Register that server and create a server endpoint for the first server to be migrated. Do not enable cloud tiering.
4. Let all of the data sync to your Azure file share (cloud endpoint).
5. Install and register the Azure File Sync agent on each of the remaining DFS-R servers.
6. Disable DFS-R.

7. Create a server endpoint on each of the DFS-R servers. Do not enable cloud tiering.
8. Ensure sync completes and test your topology as desired.
9. Retire DFS-R.
10. Cloud tiering may now be enabled on any server endpoint as desired.

For more information, see [Azure File Sync interop with Distributed File System \(DFS\)](#).

Next steps

- [Add or remove an Azure File Sync Server Endpoint](#)
- [Register or unregister a server with Azure File Sync](#)
- [Monitor Azure File Sync](#)

Azure File Sync proxy and firewall settings

1/8/2020 • 7 minutes to read • [Edit Online](#)

Azure File Sync connects your on-premises servers to Azure Files, enabling multi-site synchronization and cloud tiering features. As such, an on-premises server must be connected to the internet. An IT admin needs to decide the best path for the server to reach into Azure cloud services.

This article will provide insight into specific requirements and options available to successfully and securely connect your server to Azure File Sync.

Overview

Azure File Sync acts as an orchestration service between your Windows Server, your Azure file share, and several other Azure services to sync data as described in your sync group. For Azure File Sync to work correctly, you will need to configure your servers to communicate with the following Azure services:

- Azure Storage
- Azure File Sync
- Azure Resource Manager
- Authentication services

NOTE

The Azure File Sync agent on Windows Server initiates all requests to cloud services which results in only having to consider outbound traffic from a firewall perspective.

No Azure service initiates a connection to the Azure File Sync agent.

Ports

Azure File Sync moves file data and metadata exclusively over HTTPS and requires port 443 to be open outbound. As a result all traffic is encrypted.

Networks and special connections to Azure

The Azure File Sync agent has no requirements regarding special channels like [ExpressRoute](#), etc. to Azure.

Azure File Sync will work through any means available that allow reach into Azure, automatically adapting to various network characteristics like bandwidth, latency as well as offering admin control for fine-tuning. Not all features are available at this time. If you would like to configure specific behavior, let us know via [Azure Files UserVoice](#).

Proxy

Azure File Sync supports app-specific and machine-wide proxy settings.

App-specific proxy settings allow configuration of a proxy specifically for Azure File Sync traffic. App-specific proxy settings are supported on agent version 4.0.1.0 or newer and can be configured during the agent installation or by using the `Set-StorageSyncProxyConfiguration` PowerShell cmdlet.

PowerShell commands to configure app-specific proxy settings:

```
Import-Module "C:\Program Files\Azure\StorageSyncAgent\StorageSync.Management.ServerCmdlets.dll"
Set-StorageSyncProxyConfiguration -Address <url> -Port <port number> -ProxyCredential <credentials>
```

Machine-wide proxy settings are transparent to the Azure File Sync agent as the entire traffic of the server is routed through the proxy.

To configure machine-wide proxy settings, follow the steps below:

1. Configure proxy settings for .NET applications

- Edit these two files:
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Config\machine.config
C:\Windows\Microsoft.NET\Framework\v4.0.30319\Config\machine.config
- Add the <system.net> section in the machine.config files (below the <system.serviceModel> section). Change 127.0.0.1:8888 to the IP address and port for the proxy server.

```
<system.net>
  <defaultProxy enabled="true" useDefaultCredentials="true">
    <proxy autoDetect="false" bypassOnLocal="false" proxyAddress="http://127.0.0.1:8888"
    usesystemDefault="false" />
  </defaultProxy>
</system.net>
```

2. Set the WinHTTP proxy settings

- Run the following command from an elevated command prompt or PowerShell to see the existing proxy setting:

```
netsh winhttp show proxy
```

- Run the following command from an elevated command prompt or PowerShell to set the proxy setting (change 127.0.0.1:8888 to the IP address and port for the proxy server):

```
netsh winhttp set proxy 127.0.0.1:8888
```

3. Restart the Storage Sync Agent service by running the following command from an elevated command prompt or PowerShell:

```
net stop filesyncsvc
```

Note: The Storage Sync Agent (filesyncsvc) service will auto-start once stopped.

Firewall

As mentioned in a previous section, port 443 needs to be open outbound. Based on policies in your datacenter, branch or region, further restricting traffic over this port to specific domains may be desired or required.

The following table describes the required domains for communication:

SERVICE	PUBLIC CLOUD ENDPOINT	AZURE GOVERNMENT ENDPOINT	USAGE
Azure Resource Manager	https://management.azure.com	https://management.usgovcloudapi.net	Any user call (like PowerShell) goes to/through this URL, including the initial server registration call.

Service	Public Cloud Endpoint	Azure Government Endpoint	Usage
Azure Active Directory	https://login.windows.net https://login.microsoftonline.com	https://login.microsoftonline.us	Azure Resource Manager calls must be made by an authenticated user. To succeed, this URL is used for user authentication.
Azure Active Directory	https://graph.windows.net/	https://graph.windows.net/	As part of deploying Azure File Sync, a service principal in the subscription's Azure Active Directory will be created. This URL is used for that. This principal is used for delegating a minimal set of rights to the Azure File Sync service. The user performing the initial setup of Azure File Sync must be an authenticated user with subscription owner privileges.
Azure Storage	*.core.windows.net	*.core.usgovcloudapi.net	When the server downloads a file, then the server performs that data movement more efficiently when talking directly to the Azure file share in the Storage Account. The server has a SAS key that only allows for targeted file share access.
Azure File Sync	*.one.microsoft.com *.afs.azure.net	*.afs.azure.us	After initial server registration, the server receives a regional URL for the Azure File Sync service instance in that region. The server can use the URL to communicate directly and efficiently with the instance handling its sync.
Microsoft PKI	https://www.microsoft.com/pki/mscorp/cps http://ocsp.msocsp.com	https://www.microsoft.com/pki/mscorp/cps http://ocsp.msocsp.com	Once the Azure File Sync agent is installed, the PKI URL is used to download intermediate certificates required to communicate with the Azure File Sync service and Azure file share. The OCSP URL is used to check the status of a certificate.

IMPORTANT

When allowing traffic to *.one.microsoft.com, traffic to more than just the sync service is possible from the server. There are many more Microsoft services available under subdomains.

If *.one.microsoft.com is too broad, you can limit the server's communication by allowing communication to only explicit regional instances of the Azure Files Sync service. Which instance(s) to choose depends on the region of the storage sync service you have deployed and registered the server to. That region is called "Primary endpoint URL" in the table below.

For business continuity and disaster recovery (BCDR) reasons you may have specified your Azure file shares in a globally redundant (GRS) storage account. If that is the case, then your Azure file shares will fail over to the paired region in the event of a lasting regional outage. Azure File Sync uses the same regional pairings as storage. So if you use GRS storage accounts, you need to enable additional URLs to allow your server to talk to the paired region for Azure File Sync. The table below calls this "Paired region". Additionally, there is a traffic manager profile URL that needs to be enabled as well. This will ensure network traffic can be seamlessly re-routed to the paired region in the event of a fail-over and is called "Discovery URL" in the table below.

CLOUD	REGION	PRIMARY ENDPOINT URL	PAIRED REGION	DISCOVERY URL
Public	Australia East	https://kailani-aue.one.microsoft.com	Australia Southeast	https://tm-kailani-aue.one.microsoft.com
Public	Australia Southeast	https://kailani-aus.one.microsoft.com	Australia East	https://tm-kailani-aus.one.microsoft.com
Public	Brazil South	https://brazilsouth01.afs.azure.net	South Central US	https://tm-brazilsouth01.afs.azure.net
Public	Canada Central	https://kailani-cac.one.microsoft.com	Canada East	https://tm-kailani-cac.one.microsoft.com
Public	Canada East	https://kailani-cae.one.microsoft.com	Canada Central	https://tm-kailani.cae.one.microsoft.com
Public	Central India	https://kailani-cin.one.microsoft.com	South India	https://tm-kailani-cin.one.microsoft.com
Public	Central US	https://kailani-cus.one.microsoft.com	East US 2	https://tm-kailani-cus.one.microsoft.com
Public	East Asia	https://kailani11.one.microsoft.com	Southeast Asia	https://tm-kailani11.one.microsoft.com
Public	East US	https://kailani1.one.microsoft.com	West US	https://tm-kailani1.one.microsoft.com
Public	East US 2	https://kailani-ess.one.microsoft.com	Central US	https://tm-kailani-ess.one.microsoft.com
Public	Japan East	https://japaneast01_afs.azure.net	Japan West	https://tm-japaneast01_afs.azure.net

CLOUD	REGION	PRIMARY ENDPOINT URL	PAIRED REGION	DISCOVERY URL
Public	Japan West	https://japanwest01.efs.azure.net	Japan East	https://tm-japanwest01.efs.azure.net
Public	Korea Central	https://koreacentral01.efs.azure.net/	Korea South	https://tm-koreacentral01.efs.azure.net/
Public	Korea South	https://koreasouth01.efs.azure.net/	Korea Central	https://tm-koreasouth01.efs.azure.net/
Public	North Central US	https://northcentralus01.efs.azure.net	South Central US	https://tm-northcentralus01.efs.azure.net
Public	North Europe	https://kailani7.one.microsoft.com	West Europe	https://tm-kailani7.one.microsoft.com
Public	South Central US	https://southcentralus01.efs.azure.net	North Central US	https://tm-southcentralus01.efs.azure.net
Public	South India	https://kailani-sin.one.microsoft.com	Central India	https://tm-kailani-sin.one.microsoft.com
Public	Southeast Asia	https://kailani10.one.microsoft.com	East Asia	https://tm-kailani10.one.microsoft.com
Public	UK South	https://kailani-uks.one.microsoft.com	UK West	https://tm-kailani-uks.one.microsoft.com
Public	UK West	https://kailani-ukw.one.microsoft.com	UK South	https://tm-kailani-ukw.one.microsoft.com
Public	West Central US	https://westcentralus01.efs.azure.net	West US 2	https://tm-westcentralus01.efs.azure.net
Public	West Europe	https://kailani6.one.microsoft.com	North Europe	https://tm-kailani6.one.microsoft.com
Public	West US	https://kailani.one.microsoft.com	East US	https://tm-kailani.one.microsoft.com
Public	West US 2	https://westus201.efs.azure.net	West Central US	https://tm-westus201.efs.azure.net

CLOUD	REGION	PRIMARY ENDPOINT URL	PAIRED REGION	DISCOVERY URL
Government	US Gov Arizona	https://usgovarizona01.efs.azure.us	US Gov Texas	https://tm-usgovarizona01.efs.azure.us
Government	US Gov Texas	https://usgovtexas01.efs.azure.us	US Gov Arizona	https://tm-usgovtexas01.efs.azure.us

- If you use locally redundant (LRS) or zone redundant (ZRS) storage accounts, you only need to enable the URL listed under "Primary endpoint URL".
- If you use globally redundant (GRS) storage accounts, enable three URLs.

Example: You deploy a storage sync service in ["West US"](#) and register your server with it. The URLs to allow the server to communicate to for this case are:

- <https://kailani.one.microsoft.com> (primary endpoint: West US)
- <https://kailani1.one.microsoft.com> (paired fail-over region: East US)
- <https://tm-kailani.one.microsoft.com> (discovery URL of the primary region)

Test network connectivity to service endpoints

Once a server is registered with the Azure File Sync service, the `Test-StorageSyncNetworkConnectivity` cmdlet and `ServerRegistration.exe` can be used to test communications with all endpoints (URLs) specific to this server. This cmdlet can help troubleshoot when incomplete communication prevents the server from fully working with Azure File Sync and it can be used to fine tune proxy and firewall configurations.

To run the network connectivity test, install Azure File Sync agent version 9.1 or later and run the following PowerShell commands:

```
Import-Module "C:\Program Files\Azure\StorageSyncAgent\StorageSync.Management.ServerCmdlets.dll"
Test-StorageSyncNetworkConnectivity
```

Summary and risk limitation

The lists earlier in this document contain the URLs Azure File Sync currently communicates with. Firewalls must be able to allow traffic outbound to these domains. Microsoft strives to keep this list updated.

Setting up domain restricting firewall rules can be a measure to improve security. If these firewall configurations are used, one needs to keep in mind that URLs will be added and might even change over time. Check this article periodically.

Next steps

- [Planning for an Azure File Sync deployment](#)
- [Deploy Azure File Sync](#)
- [Monitor Azure File Sync](#)

Configure a SQL Server failover cluster instance with premium file share on Azure virtual machines

2/26/2020 • 19 minutes to read • [Edit Online](#)

This article explains how to create a SQL Server failover cluster instance (FCI) on Azure virtual machines by using a [premium file share](#).

Premium file shares are SSD-backed, consistently low-latency file shares that are fully supported for use with Failover Cluster Instances for SQL Server 2012 or later on Windows Server 2012 or later. Premium file shares give you greater flexibility, allowing you to resize and scale a file share without any downtime.

Before you begin

There are a few things you need to know and have in place before you start.

You should have an operational understanding of these technologies:

- [Windows cluster technologies](#)
- [SQL Server Failover Cluster Instances](#)

One thing to be aware of is that on an Azure IaaS VM failover cluster, we recommend a single NIC per server (cluster node) and a single subnet. Azure networking has physical redundancy that makes additional NICs and subnets unnecessary on an Azure IaaS VM guest cluster. The cluster validation report will warn you that the nodes are reachable only on a single network. You can ignore this warning on Azure IaaS VM failover clusters.

You should also have a general understanding of these technologies:

- [Azure premium file share](#)
- [Azure resource groups](#)

IMPORTANT

At this time, SQL Server failover cluster instances on Azure virtual machines are only supported with the [lightweight management mode](#) of the [SQL Server IaaS Agent Extension](#). To change from full extension mode to lightweight, delete the **SQL Virtual Machine** resource for the corresponding VMs and then register them with the SQL VM resource provider in lightweight mode. When deleting the **SQL Virtual Machine** resource using the Azure portal, **clear the checkbox next to the correct Virtual Machine**. The full extension supports features such as automated backup, patching, and advanced portal management. These features will not work for SQL VMs after the agent is reinstalled in lightweight management mode.

Premium file shares provide IOPS and throughout capacities that will meet the needs of many workloads. For IO-intensive workloads, consider [SQL Server Failover Cluster Instances with Storage Spaces Direct](#), based on managed premium disks or ultra disks.

Check the IOPS activity of your environment and verify that premium file shares will provide the IOPS you need before you start a deployment or migration. Use Windows Performance Monitor disk counters to monitor the total IOPS (Disk Transfers/second) and throughput (Disk Bytes/second) required for SQL Server Data, Log, and Temp DB files.

Many workloads have bursting IO, so it's a good idea to check during heavy usage periods and note both the maximum IOPS and the average IOPS. Premium file shares provide IOPS based on the size of the share. Premium file shares also provide complimentary bursting that allows you to burst your IO to triple the baseline amount for

up to one hour.

For more information about premium file share performance, see [File share performance tiers](#).

Licensing and pricing

On Azure virtual machines, you can license SQL Server by using pay-as-you-go (PAYG) or bring-your-own-license (BYOL) VM images. The type of image you choose affects how you're charged.

With pay-as-you-go licensing, a failover cluster instance (FCI) of SQL Server on Azure virtual machines incurs charges for all nodes of the FCI, including the passive nodes. For more information, see [SQL Server Enterprise Virtual Machines Pricing](#).

If you have Enterprise Agreement with Software Assurance, you can use one free passive FCI node for each active node. To take advantage of this benefit in Azure, use BYOL VM images, and use the same license on both the active and passive nodes of the FCI. For more information, see [Enterprise Agreement](#).

To compare pay-as-you-go and BYOL licensing for SQL Server on Azure virtual machines, see [Get started with SQL VMs](#).

For complete information about licensing SQL Server, see [Pricing](#).

Filestream

Filestream isn't supported for a failover cluster with a premium file share. To use filestream, deploy your cluster by using [Storage Spaces Direct](#).

Prerequisites

Before you complete the steps in this article, you should already have:

- A Microsoft Azure subscription.
- A Windows domain on Azure virtual machines.
- A domain user account that has permissions to create objects on both Azure virtual machines and in Active Directory.
- A domain user account to run the SQL Server service and that you can log into the virtual machine with when mounting the file share.
- An Azure virtual network and subnet with enough IP address space for these components:
 - Two virtual machines.
 - The failover cluster IP address.
 - An IP address for each FCI.
- DNS configured on the Azure network, pointing to the domain controllers.
- A [premium file share](#) to be used as the clustered drive, based on the storage quota of your database for your data files.
- If you're on Windows Server 2012 R2 and older, you will need another file share to use as the file share witness, since cloud witnesses are supported for Windows 2016 and newer. You can use another Azure file share, or you can use a file share on a separate virtual machine. If you're going to use another Azure file share, you can mount it with the same process as for the premium file share used for your clustered drive.

With these prerequisites in place, you can start building your failover cluster. The first step is to create the virtual machines.

Step 1: Create the virtual machines

1. Sign in to the [Azure portal](#) with your subscription.
2. [Create an Azure availability set](#).

The availability set groups virtual machines across fault domains and update domains. It ensures that your application isn't affected by single points of failure, like the network switch or the power unit of a rack of servers.

If you haven't created the resource group for your virtual machines, do it when you create an Azure availability set. If you're using the Azure portal to create the availability set, take these steps:

- a. In the Azure portal, select **Create a resource** to open Azure Marketplace. Search for **Availability set**.
- b. Select **Availability Set**.
- c. Select **Create**.
- d. Under **Create availability set**, provide these values:
 - **Name**: A name for the availability set.
 - **Subscription**: Your Azure subscription.
 - **Resource group**: If you want to use an existing group, click **Select existing** and then select the group from the list. Otherwise, select **Create new** and enter a name for the group.
 - **Location**: Set the location where you plan to create your virtual machines.
 - **Fault domains**: Use the default (3).
 - **Update domains**: Use the default (5).
- e. Select **Create** to create the availability set.

3. Create the virtual machines in the availability set.

Provision two SQL Server virtual machines in the Azure availability set. For instructions, see [Provision a SQL Server virtual machine in the Azure portal](#).

Place both virtual machines:

- In the same Azure resource group as your availability set.
- On the same network as your domain controller.
- On a subnet that has sufficient IP address space for both virtual machines and all FCIs that you might eventually use on the cluster.
- In the Azure availability set.

IMPORTANT

You can't set or change the availability set after you've created a virtual machine.

Choose an image from Azure Marketplace. You can use an Azure Marketplace image that includes Windows Server and SQL Server, or use one that just includes Windows Server. For details, see [Overview of SQL Server on Azure virtual machines](#).

The official SQL Server images in the Azure Gallery include an installed SQL Server instance, the SQL Server installation software, and the required key.

IMPORTANT

After you create the virtual machine, remove the pre-installed standalone SQL Server instance. You'll use the pre-installed SQL Server media to create the SQL Server FCI after you set up the failover cluster and premium file share as storage.

Alternatively, you can use Azure Marketplace images that contain just the operating system. Choose a **Windows Server 2016 Datacenter** image and install the SQL Server FCI after you set up the failover

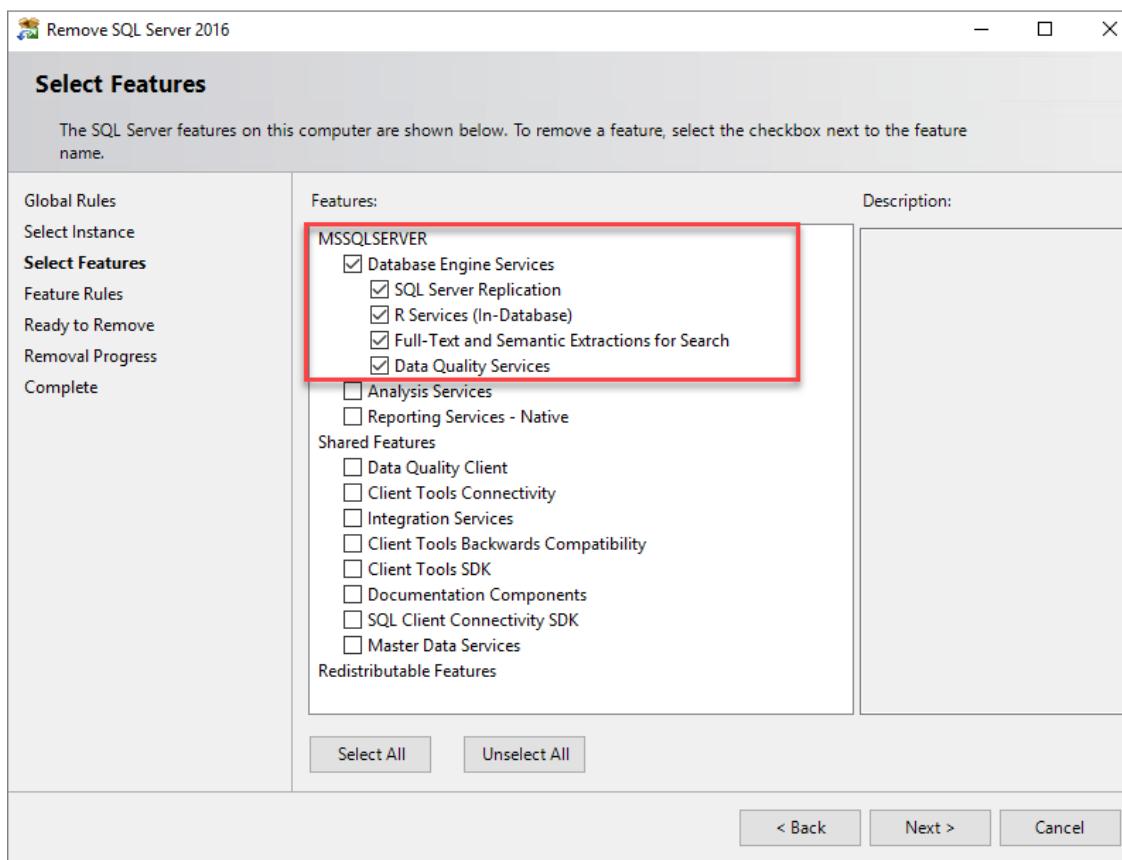
cluster and premium file share as storage. This image doesn't contain SQL Server installation media. Place the SQL Server installation media in a location where you can run it for each server.

- After Azure creates your virtual machines, connect to each one by using RDP.

When you first connect to a virtual machine by using RDP, a prompt asks you if you want to allow the PC to be discoverable on the network. Select **Yes**.

- If you're using one of the SQL Server-based virtual machine images, remove the SQL Server instance.

- In **Programs and Features**, right-click **Microsoft SQL Server 201_ (64-bit)** and select **Uninstall/Change**.
- Select **Remove**.
- Select the default instance.
- Remove all features under **Database Engine Services**. Don't remove **Shared Features**. You'll see something like the following screenshot:



- Select **Next**, and then select **Remove**.

- Open the firewall ports.

On each virtual machine, open these ports on the Windows Firewall:

PURPOSE	TCP PORT	NOTES
SQL Server	1433	Normal port for default instances of SQL Server. If you used an image from the gallery, this port is automatically opened.

PURPOSE	TCP PORT	NOTES
Health probe	59999	Any open TCP port. In a later step, configure the load balancer health probe and the cluster to use this port.
File share	445	Port used by the file share service.

7. Add the virtual machines to your pre-existing domain.

After you create and configure the virtual machines, you can configure the premium file share.

Step 2: Mount the premium file share

1. Sign in to the [Azure portal](#) and go to your storage account.
2. Go to **File Shares** under **File service** and select the premium file share you want to use for your SQL storage.
3. Select **Connect** to bring up the connection string for your file share.
4. Select the drive letter you want to use from the drop-down list and then copy both code blocks to Notepad.

The screenshot shows the Azure Storage Account Overview page for 'sqlpremiumfileshare'. On the left, there's a sidebar with 'Settings' and options like 'Access policy' and 'Properties'. The main area shows the file share name 'sqlpremiumfileshare' and a 'Connect' button highlighted with a red box. Below it, there's a note about secure transfer and a 'Backup (Preview)' button. The 'Location' is listed as 'sqlpremi'. On the right, there are tabs for 'Windows', 'Linux', and 'MacOS'. Under 'Windows', a 'Drive letter' dropdown is set to 'G' and highlighted with a red box. Below it, a code block shows PowerShell commands to connect from a Windows computer. An alternative command for non-forward-slash users is also provided. A note at the bottom about opening port 445 is also highlighted with a red box.

5. Use RDP to connect to the SQL Server VM with the account that your SQL Server FCI will use for the service account.
6. Open an administrative PowerShell command console.
7. Run the commands that you saved earlier when you were working in the portal.

8. Go to the share by using either File Explorer or the **Run** dialog box (Windows logo key + r). Use the network path `\storageaccountname.file.core.windows.net\filesharename`. For example,
`\sqlvmstorageaccount.file.core.windows.net\sqlpremiumfileshare`
9. Create at least one folder on the newly connected file share to place your SQL Data files into.
10. Repeat these steps on each SQL Server VM that will participate in the cluster.

IMPORTANT

- Consider using a separate file share for backup files to save the IOPS and space capacity of this share for Data and Log files. You can use either a premium or standard file share for backup files.
- If you're on Windows 2012 R2 and older, follow these same steps to mount your file share that you are going to use as the file share witness.

Step 3: Configure the failover cluster

The next step is to configure the failover cluster. In this step, you'll complete the following substeps:

1. Add the Windows Server Failover Clustering feature.
2. Validate the cluster.
3. Create the failover cluster.
4. Create the cloud witness (for Windows Server 2016 and newer) or the file share witness (for Windows Server 2012 R2 and older).

Add Windows Server Failover Clustering

1. Connect to the first virtual machine with RDP by using a domain account that's a member of the local administrators and that has permission to create objects in Active Directory. Use this account for the rest of the configuration.
2. [Add Failover Clustering to each virtual machine.](#)

To install Failover Clustering from the UI, take these steps on both virtual machines:

- a. In **Server Manager**, select **Manage**, and then select **Add Roles and Features**.
- b. In the **Add Roles and Features Wizard**, select **Next** until you get to **Select Features**.
- c. In **Select Features**, select **Failover Clustering**. Include all required features and the management tools.
 Select **Add Features**.
- d. Select **Next**, and then select **Finish** to install the features.

To install Failover Clustering by using PowerShell, run the following script from an administrator PowerShell session on one of the virtual machines:

```
$nodes = ("<node1>","<node2>")
Invoke-Command $nodes {Install-WindowsFeature Failover-Clustering -IncludeAllSubFeature -IncludeManagementTools}
```

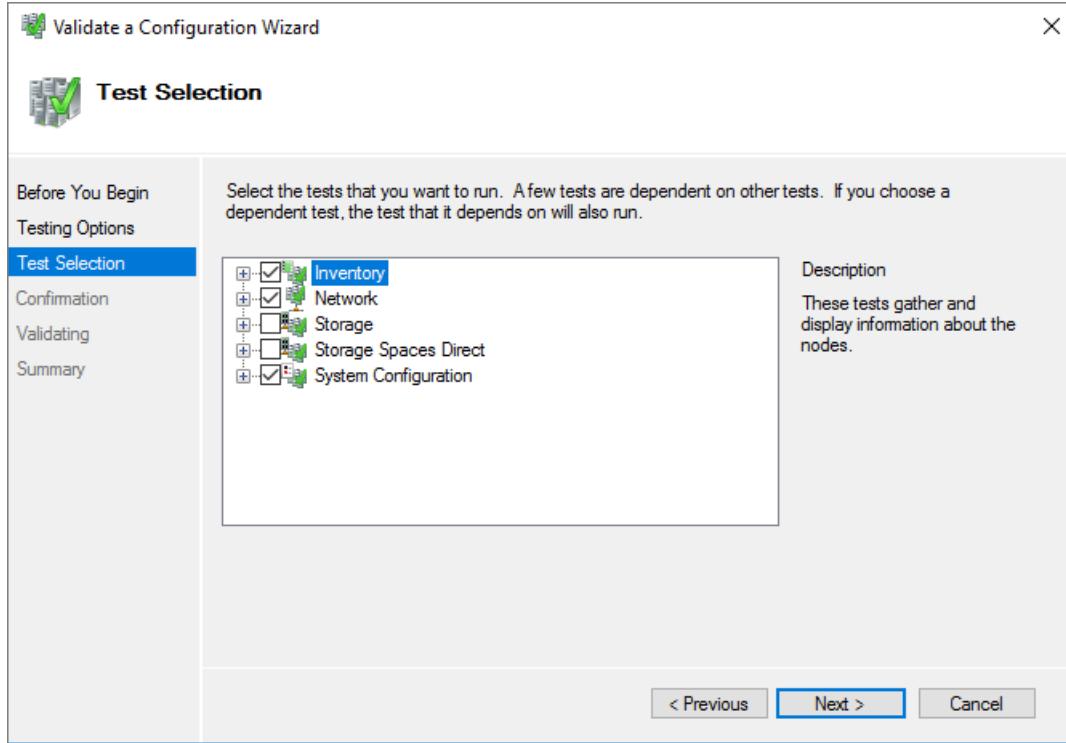
Validate the cluster

Validate the cluster in the UI or by using PowerShell.

To validate the cluster by using the UI, take the following steps on one of the virtual machines:

1. Under **Server Manager**, select **Tools**, and then select **Failover Cluster Manager**.
2. Under **Failover Cluster Manager**, select **Action**, and then select **Validate Configuration**.

3. Select **Next**.
4. Under **Select Servers or a Cluster**, enter the names of both virtual machines.
5. Under **Testing options**, select **Run only tests I select**. Select **Next**.
6. Under **Test Selection**, select all tests except for **Storage** and **Storage Spaces Direct**, as shown here:



7. Select **Next**.
8. Under **Confirmation**, select **Next**.

The **Validate a Configuration Wizard** runs the validation tests.

To validate the cluster by using PowerShell, run the following script from an administrator PowerShell session on one of the virtual machines:

```
Test-Cluster -Node ("<node1>","<node2>") -Include "Inventory", "Network", "System Configuration"
```

After you validate the cluster, create the failover cluster.

Create the failover cluster

To create the failover cluster, you need:

- The names of the virtual machines that will become the cluster nodes.
- A name for the failover cluster
- An IP address for the failover cluster. You can use an IP address that's not used on the same Azure virtual network and subnet as the cluster nodes.

Windows Server 2012 through Windows Server 2016

The following PowerShell script creates a failover cluster for Windows Server 2012 through Windows Server 2016. Update the script with the names of the nodes (the virtual machine names) and an available IP address from the Azure virtual network.

```
New-Cluster -Name <FailoverCluster-Name> -Node ("<node1>","<node2>") -StaticAddress <n.n.n.n> -NoStorage
```

Windows Server 2019

The following PowerShell script creates a failover cluster for Windows Server 2019. For more information, see [Failover cluster: Cluster Network Object](#). Update the script with the names of the nodes (the virtual machine names) and an available IP address from the Azure virtual network.

```
New-Cluster -Name <FailoverCluster-Name> -Node ("<node1>","<node2>") -StaticAddress <n.n.n.n> -NoStorage -ManagementPointNetworkType Singleton
```

Create a cloud witness (Win 2016 +)

If you're on Windows Server 2016 and greater, you'll need to create a Cloud Witness. Cloud Witness is a new type of cluster quorum witness that's stored in an Azure storage blob. This removes the need for a separate VM that hosts a witness share, or using a separate file share.

1. [Create a cloud witness for the failover cluster](#).
2. Create a blob container.
3. Save the access keys and the container URL.

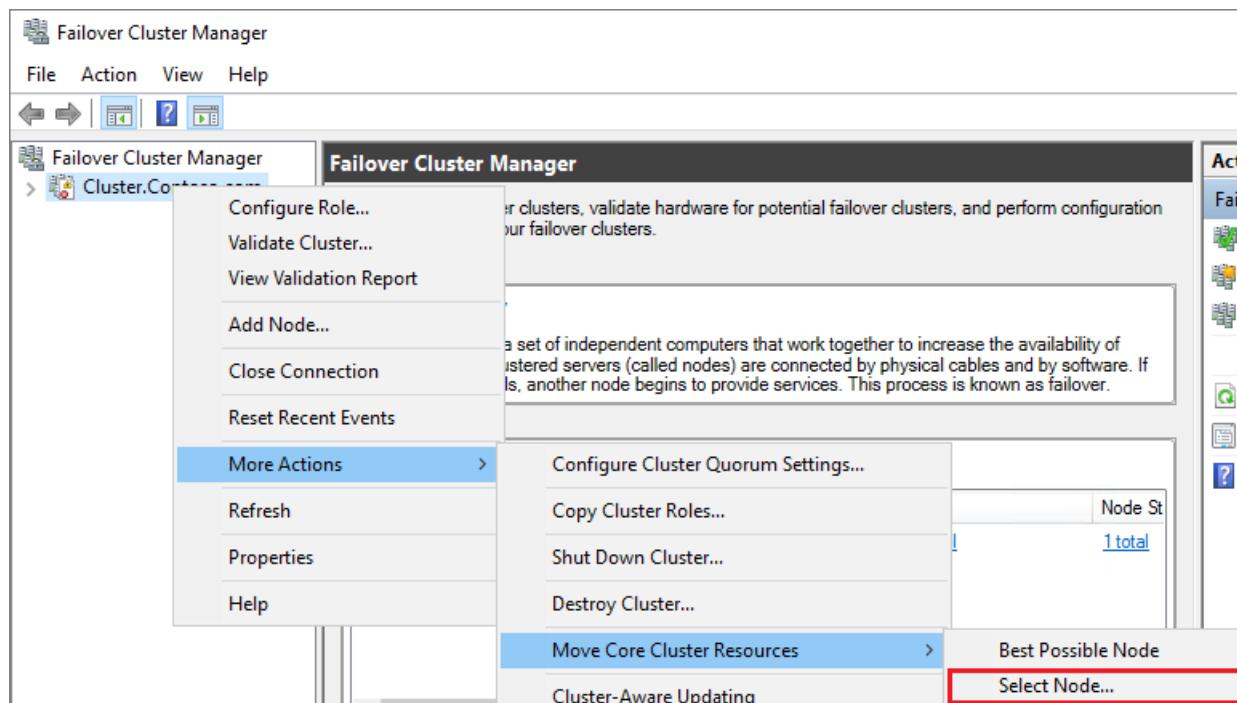
Configure quorum

For Windows Server 2016 and greater, configure the cluster to use the cloud witness you just created. Follow all of the steps [Configure the quorum witness in the user interface](#).

For Windows Server 2012 R2 and older, follow the same steps in [Configure the quorum witness in the user interface](#) but on the **Select Quorum Witness** page, select the **Configure a file share witness** option. Specify the file share you allocated to be the file share witness, whether it's one you configured on a separate virtual machine, or mounted from Azure.

Step 4: Test cluster failover

Test failover of your cluster. In **Failover Cluster Manager**, right-click your cluster and select **More Actions > Move Core Cluster Resource > Select node**, and then select the other node of the cluster. Move the core cluster resource to every node of the cluster, and then move it back to the primary node. If you can successfully move the cluster to each node, you're ready to install SQL Server.

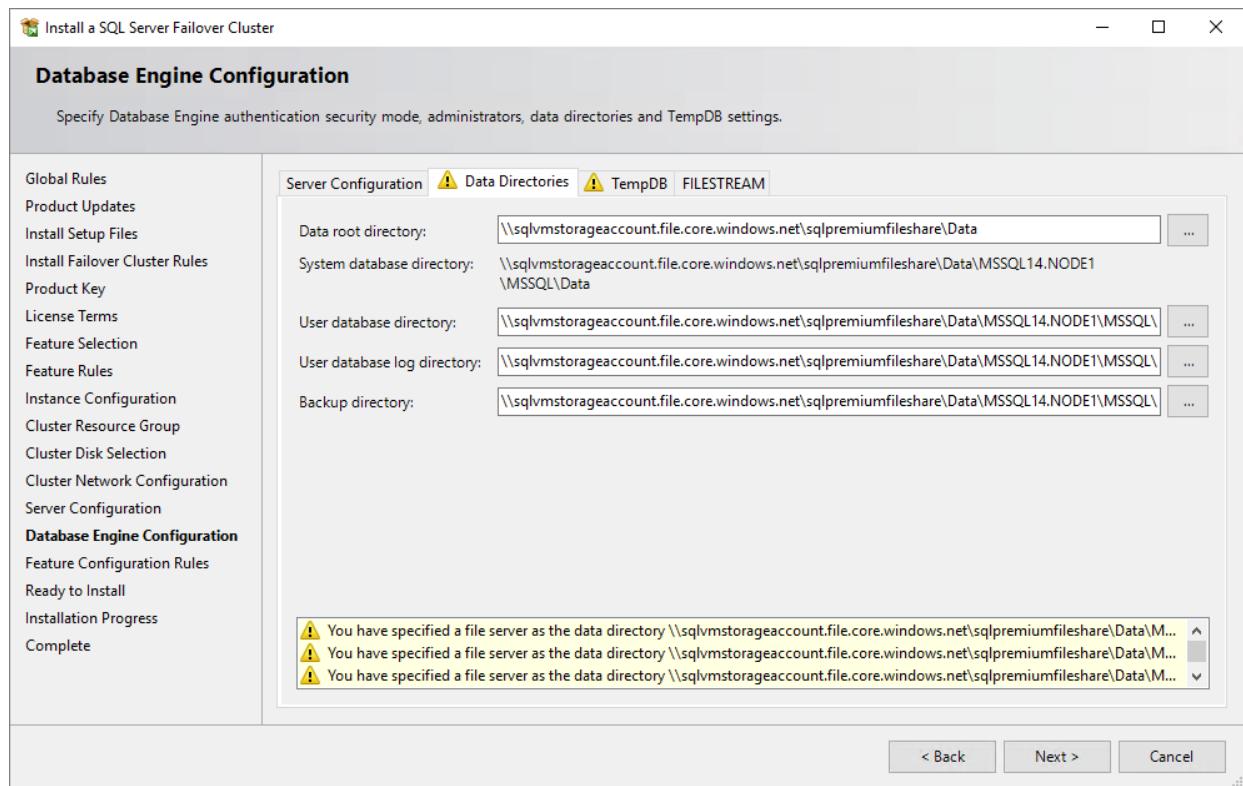


Step 5: Create the SQL Server FCI

After you've configured the failover cluster, you can create the SQL Server FCI.

1. Connect to the first virtual machine by using RDP.
2. In **Failover Cluster Manager**, make sure all the Core Cluster Resources are on the first virtual machine. If you need to, move all resources to this virtual machine.
3. Locate the installation media. If the virtual machine uses one of the Azure Marketplace images, the media is located at `C:\SQLServer_<version number>_Full`. Select **Setup**.
4. In the **SQL Server Installation Center**, select **Installation**.
5. Select **New SQL Server failover cluster installation**. Follow the instructions in the wizard to install the SQL Server FCI.

The FCI data directories need to be on the premium file share. Enter the full path of the share, in this form: `\storageaccountname.file.core.windows.net\filesharename\foldername`. A warning will appear, telling you that you've specified a file server as the data directory. This warning is expected. Ensure that the user account you RDP'd into the VM with when you persisted the file share is the same account that the SQL Server service uses to avoid possible failures.



6. After you complete the steps in the wizard, Setup will install a SQL Server FCI on the first node.
7. After Setup installs the FCI on the first node, connect to the second node by using RDP.
8. Open the **SQL Server Installation Center**. Select **Installation**.
9. Select **Add node to a SQL Server failover cluster**. Follow the instructions in the wizard to install SQL Server and add the server to the FCI.

NOTE

If you used an Azure Marketplace gallery image with SQL Server, SQL Server tools were included with the image. If you didn't use one of those images, install the SQL Server tools separately. See [Download SQL Server Management Studio \(SSMS\)](#).

Step 6: Create the Azure load balancer

On Azure virtual machines, clusters use a load balancer to hold an IP address that needs to be on one cluster node at a time. In this solution, the load balancer holds the IP address for the SQL Server FCI.

For more information, see [Create and configure an Azure load balancer](#).

Create the load balancer in the Azure portal

To create the load balancer:

1. In the Azure portal, go to the resource group that contains the virtual machines.
2. Select **Add**. Search the Azure Marketplace for **Load Balancer**. Select **Load Balancer**.
3. Select **Create**.
4. Set up the load balancer by using the following values:
 - **Subscription**: Your Azure subscription.
 - **Resource group**: The resource group that contains your virtual machines.
 - **Name**: A name that identifies the load balancer.
 - **Region**: The Azure location that contains your virtual machines.
 - **Type**: Either public or private. A private load balancer can be accessed from within the virtual network.
Most Azure applications can use a private load balancer. If your application needs access to SQL Server directly over the internet, use a public load balancer.
 - **SKU**: Standard.
 - **Virtual network**: The same network as the virtual machines.
 - **IP address assignment**: Static.
 - **Private IP address**: The IP address that you assigned to the SQL Server FCI cluster network resource.

The following image shows the **Create load balancer** UI:

Create load balancer

[Basics](#) [Tags](#) [Review + create](#)

Azure load balancer is a layer 4 load balancer that distributes incoming traffic among healthy virtual machine instances. Load balancers uses a hash-based distribution algorithm. By default, it uses a 5-tuple (source IP, source port, destination IP, destination port, protocol type) hash to map traffic to available servers. Load balancers can either be internet-facing where it is accessible via public IP addresses, or internal where it is only accessible from a virtual network. Azure load balancers also support Network Address Translation (NAT) to route traffic between public and private IP addresses. [Learn more.](#)

Project details

Subscription *

Resource group *

[Create new](#)

Instance details

Name *



Region *



Type * ⓘ

Internal Public

SKU * ⓘ

Basic Standard

Configure virtual network.

Virtual network * ⓘ



IP address assignment *

Static Dynamic

Private IP address *



Configure the load balancer backend pool

1. Return to the Azure resource group that contains the virtual machines and locate the new load balancer. You might need to refresh the view on the resource group. Select the load balancer.
2. Select **Backend pools**, and then select **Add**.
3. Associate the backend pool with the availability set that contains the VMs.
4. Under **Target network IP configurations**, select **VIRTUAL MACHINE** and choose the virtual machines that will participate as cluster nodes. Be sure to include all virtual machines that will host the FCI.
5. Select **OK** to create the backend pool.

Configure a load balancer health probe

1. On the load balancer blade, select **Health probes**.
2. Select **Add**.
3. On the **Add health probe** blade, set the following health probe parameters.
 - **Name:** A name for the health probe.
 - **Protocol:** TCP.
 - **Port:** The port you created in the firewall for the health probe in [this step](#). In this article, the example uses TCP port **59999**.
 - **Interval:** 5 Seconds.
 - **Unhealthy threshold:** 2 consecutive failures.

4. Select **OK**.

Set load balancing rules

1. On the load balancer blade, select **Load balancing rules**.

2. Select **Add**.

3. Set the load balancing rule parameters:

- **Name**: A name for the load balancing rules.
- **Frontend IP address**: The IP address for the SQL Server FCI cluster network resource.
- **Port**: The SQL Server FCI TCP port. The default instance port is 1433.
- **Backend port**: Uses the same port as the **Port** value when you enable **Floating IP (direct server return)**.
- **Backend pool**: The backend pool name that you configured earlier.
- **Health probe**: The health probe that you configured earlier.
- **Session persistence**: None.
- **Idle timeout (minutes)**: 4.
- **Floating IP (direct server return)**: Enabled.

4. Select **OK**.

Step 7: Configure the cluster for the probe

Set the cluster probe port parameter in PowerShell.

To set the cluster probe port parameter, update the variables in the following script with values from your environment. Remove the angle brackets (< and >) from the script.

```
$ClusterNetworkName = "<Cluster Network Name>"  
$IPResourceName = "<SQL Server FCI IP Address Resource Name>"  
$ILBIP = "<n.n.n.n>"  
[int]$ProbePort = <nnnnn>  
  
Import-Module FailoverClusters  
  
Get-ClusterResource $IPResourceName | Set-ClusterParameter -Multiple  
@{"Address"="$ILBIP"; "ProbePort"=$ProbePort; "SubnetMask"="255.255.255.255"; "Network"="$ClusterNetworkName"; "EnabledDhcp"=0}
```

The following list describes the values that you need to update:

- <Cluster Network Name> : The Windows Server Failover Cluster name for the network. In **Failover Cluster Manager** > **Networks**, right-click the network and select **Properties**. The correct value is under **Name** on the **General** tab.
- <SQL Server FCI IP Address Resource Name> : The SQL Server FCI IP address resource name. In **Failover Cluster Manager** > **Roles**, under the SQL Server FCI role, under **Server Name**, right-click the IP address resource and select **Properties**. The correct value is under **Name** on the **General** tab.
- <ILBIP> : The ILB IP address. This address is configured in the Azure portal as the ILB front-end address. This is also the SQL Server FCI IP address. You can find it in **Failover Cluster Manager** on the same properties page where you located the <SQL Server FCI IP Address Resource Name> .
- <nnnnn> : The probe port you configured in the load balancer health probe. Any unused TCP port is valid.

IMPORTANT

The subnet mask for the cluster parameter must be the TCP IP broadcast address: `255.255.255.255`.

After you set the cluster probe, you can see all the cluster parameters in PowerShell. Run this script:

```
Get-ClusterResource $IPResourceName | Get-ClusterParameter
```

Step 8: Test FCI failover

Test failover of the FCI to validate cluster functionality. Take the following steps:

1. Connect to one of the SQL Server FCI cluster nodes by using RDP.
2. Open **Failover Cluster Manager**. Select **Roles**. Notice which node owns the SQL Server FCI role.
3. Right-click the SQL Server FCI role.
4. Select **Move**, and then select **Best Possible Node**.

Failover Cluster Manager shows the role, and its resources go offline. The resources then move and come back online in the other node.

Test connectivity

To test connectivity, sign in to another virtual machine in the same virtual network. Open **SQL Server Management Studio** and connect to the SQL Server FCI name.

NOTE

If you need to, you can [download SQL Server Management Studio](#).

Limitations

Azure virtual machines support Microsoft Distributed Transaction Coordinator (MSDTC) on Windows Server 2019 with storage on Clustered Shared Volumes (CSV) and a [standard load balancer](#).

On Azure virtual machines, MSDTC isn't supported on Windows Server 2016 or earlier because:

- The clustered MSDTC resource can't be configured to use shared storage. On Windows Server 2016, if you create an MSDTC resource, it won't show any shared storage available for use, even if storage is available. This issue has been fixed in Windows Server 2019.
- The basic load balancer doesn't handle RPC ports.

See also

- [Windows cluster technologies](#)
- [SQL Server Failover Cluster Instances](#)

Use an Azure file share with Windows

2/25/2020 • 12 minutes to read • [Edit Online](#)

[Azure Files](#) is Microsoft's easy-to-use cloud file system. Azure file shares can be seamlessly used in Windows and Windows Server. This article discusses the considerations for using an Azure file share with Windows and Windows Server.

In order to use an Azure file share outside of the Azure region it is hosted in, such as on-premises or in a different Azure region, the OS must support SMB 3.0.

You can use Azure file shares on a Windows installation that is running either in an Azure VM or on-premises. The following table illustrates which OS versions support accessing file shares in which environment:

WINDOWS VERSION	SMB VERSION	MOUNTABLE IN AZURE VM	MOUNTABLE ON-PREMISES
Windows Server 2019	SMB 3.0	Yes	Yes
Windows 10 ¹	SMB 3.0	Yes	Yes
Windows Server semi-annual channel ²	SMB 3.0	Yes	Yes
Windows Server 2016	SMB 3.0	Yes	Yes
Windows 8.1	SMB 3.0	Yes	Yes
Windows Server 2012 R2	SMB 3.0	Yes	Yes
Windows Server 2012	SMB 3.0	Yes	Yes
Windows 7 ³	SMB 2.1	Yes	No
Windows Server 2008 R2 ³	SMB 2.1	Yes	No

¹Windows 10, versions 1507, 1607, 1709, 1803, 1809, 1903, and 1909.

²Windows Server, versions 1809, 1903, and 1909.

³Regular Microsoft support for Windows 7 and Windows Server 2008 R2 has ended. It is possible to purchase additional support for security updates only through the [Extended Security Update \(ESU\) program](#). We strongly recommend migrating off of these operating systems.

NOTE

We always recommend taking the most recent KB for your version of Windows.

Prerequisites

- **Storage account name:** To mount an Azure file share, you will need the name of the storage account.
- **Storage account key:** To mount an Azure file share, you will need the primary (or secondary) storage key. SAS keys are not currently supported for mounting.

- **Ensure port 445 is open:** The SMB protocol requires TCP port 445 to be open; connections will fail if port 445 is blocked. You can check to see if your firewall is blocking port 445 with the `Test-NetConnection` cmdlet. You can learn about [various ways to workaround blocked port 445 here](#).

The following PowerShell code assumes you have the Azure PowerShell module installed, see [Install Azure PowerShell module](#) for more information. Remember to replace `<your-storage-account-name>` and `<your-resource-group-name>` with the relevant names for your storage account.

```
$resourceGroupName = "<your-resource-group-name>"  
$storageAccountName = "<your-storage-account-name>"  
  
# This command requires you to be logged into your Azure account, run Login-AzAccount if you haven't  
# already logged in.  
$storageAccount = Get-AzStorageAccount -ResourceGroupName $resourceGroupName -Name  
$storageAccountName  
  
# The ComputerName, or host, is <storage-account>.file.core.windows.net for Azure Public Regions.  
# $storageAccount.Context.FileEndpoint is used because non-Public Azure regions, such as sovereign  
clouds  
# or Azure Stack deployments, will have different hosts for Azure file shares (and other storage  
resources).  
Test-NetConnection -ComputerName ([System.Uri]::new($storageAccount.Context.FileEndPoint).Host) -  
Port 445
```

If the connection was successful, you should see the following output:

```
ComputerName      : <storage-account-host-name>  
RemoteAddress    : <storage-account-ip-address>  
RemotePort       : 445  
InterfaceAlias   : <your-network-interface>  
SourceAddress    : <your-ip-address>  
TcpTestSucceeded : True
```

NOTE

The above command returns the current IP address of the storage account. This IP address is not guaranteed to remain the same, and may change at any time. Do not hardcode this IP address into any scripts, or into a firewall configuration.

Using an Azure file share with Windows

To use an Azure file share with Windows, you must either mount it, which means assigning it a drive letter or mount point path, or access it via its [UNC path](#).

Unlike other SMB shares you may have interacted with, such as those hosted on a Windows Server, Linux Samba server, or NAS device, Azure file shares do not currently support Kerberos authentication with your Active Directory (AD) or Azure Active Directory (AAD) identity, although this is a feature we are [working on](#). Instead, you must access your Azure file share with the storage account key for the storage account containing your Azure file share. A storage account key is an administrator key for a storage account, including administrator permissions to all files and folders within the file share you're accessing, and for all file shares and other storage resources (blobs, queues, tables, etc.) contained within your storage account. If this is not sufficient for your workload, [Azure File Sync](#) may address the lack of Kerberos authentication and ACL support in the interim until AAD-based Kerberos authentication and ACL support is publicly available.

A common pattern for lifting and shifting line-of-business (LOB) applications that expect an SMB file share to Azure is to use an Azure file share as an alternative for running a dedicated Windows file server in an Azure

VM. One important consideration for successfully migrating a line-of-business application to use an Azure file share is that many line-of-business applications run under the context of a dedicated service account with limited system permissions rather than the VM's administrative account. Therefore, you must ensure that you mount/save the credentials for the Azure file share from the context of the service account rather than your administrative account.

Persisting Azure file share credentials in Windows

The `cmdkey` utility allows you to store your storage account credentials within Windows. This means that when you try to access an Azure file share via its UNC path or mount the Azure file share, you will not need to specify credentials. To save your storage account's credentials, run the following PowerShell commands, replacing `<your-storage-account-name>` and `<your-resource-group-name>` where appropriate.

```
$resourceGroupName = "<your-resource-group-name>"  
$storageAccountName = "<your-storage-account-name>"  
  
# These commands require you to be logged into your Azure account, run Login-AzAccount if you haven't  
# already logged in.  
$storageAccount = Get-AzStorageAccount -ResourceGroupName $resourceGroupName -Name $storageAccountName  
$storageAccountKeys = Get-AzStorageAccountKey -ResourceGroupName $resourceGroupName -Name  
$storageAccountName  
  
# The cmdkey utility is a command-line (rather than PowerShell) tool. We use Invoke-Expression to allow us  
to  
# consume the appropriate values from the storage account variables. The value given to the add parameter  
of the  
# cmdkey utility is the host address for the storage account, <storage-account>.file.core.windows.net for  
Azure  
# Public Regions. $storageAccount.Context.FileEndpoint is used because non-Public Azure regions, such as  
sovereign  
# clouds or Azure Stack deployments, will have different hosts for Azure file shares (and other storage  
resources).  
Invoke-Expression -Command ("cmdkey /add:$([System.Uri]::new($storageAccount.Context.FileEndPoint).Host) "  
+  
"/user:AZURE\$($storageAccount.StorageAccountName) /pass:$($storageAccountKeys[0].Value)")
```

You can verify the `cmdkey` utility has stored the credential for the storage account by using the `list` parameter:

```
cmdkey /list
```

If the credentials for your Azure file share are stored successfully, the expected output is as follows (there may be additional keys stored in the list):

```
Currently stored credentials:
```

```
Target: Domain:target=<storage-account-host-name>  
Type: Domain Password  
User: AZURE\<your-storage-account-name>
```

You should now be able to mount or access the share without having to supply additional credentials.

Advanced cmdkey scenarios

There are two additional scenarios to consider with `cmdkey`: storing credentials for another user on the machine, such as a service account, and storing credentials on a remote machine with PowerShell remoting.

Storing the credentials for another user on the machine is very easy: when logged into your account, simply execute the following PowerShell command:

```
$password = ConvertTo-SecureString -String "<service-account-password>" -AsPlainText -Force
$credential = New-Object System.Management.Automation.PSCredential -ArgumentList "<service-account-username>", $password
Start-Process -FilePath PowerShell.exe -Credential $credential -LoadUserProfile
```

This will open a new PowerShell window under the user context of your service account (or user account). You can then use the cmdkey utility as described [above](#).

Storing the credentials on a remote machine using PowerShell remoting is not however possible, as cmdkey does not allow access, even for additions, to its credential store when the user is logged in via PowerShell remoting. We recommend logging into the machine with [Remote Desktop](#).

Mount the Azure file share with PowerShell

Run the following commands from a regular (i.e. not an elevated) PowerShell session to mount the Azure file share. Remember to replace `<your-resource-group-name>`, `<your-storage-account-name>`, `<your-file-share-name>`, and `<desired-drive-letter>` with the proper information.

```
$resourceGroupName = "<your-resource-group-name>"
$storageAccountName = "<your-storage-account-name>"
$fileShareName = "<your-file-share-name>

# These commands require you to be logged into your Azure account, run Login-AzAccount if you haven't
# already logged in.
$storageAccount = Get-AzStorageAccount -ResourceGroupName $resourceGroupName -Name $storageAccountName
$storageAccountKeys = Get-AzStorageAccountKey -ResourceGroupName $resourceGroupName -Name
$storageAccountName
$fileShare = Get-AzStorageShare -Context $storageAccount.Context | Where-Object {
    $_.Name -eq $fileShareName -and $_.IsSnapshot -eq $false
}

if ($fileShare -eq $null) {
    throw [System.Exception]::new("Azure file share not found")
}

# The value given to the root parameter of the New-PSDrive cmdlet is the host address for the storage
# account,
# <storage-account>.file.core.windows.net for Azure Public Regions. $fileShare.StorageUri.PrimaryUri.Host
# is
# used because non-Public Azure regions, such as sovereign clouds or Azure Stack deployments, will have
# different
# hosts for Azure file shares (and other storage resources).
$password = ConvertTo-SecureString -String $storageAccountKeys[0].Value -AsPlainText -Force
$credential = New-Object System.Management.Automation.PSCredential -ArgumentList
"AZURE\$(($storageAccount.StorageAccountName))", $password
New-PSDrive -Name <desired-drive-letter> -PSProvider FileSystem -Root
"\\"$(($fileShare.StorageUri.PrimaryUri.Host))\$($fileShare.Name)" -Credential $credential -Persist
```

NOTE

Using the `-Persist` option on the `New-PSDrive` cmdlet will only allow the file share to be remounted on boot if the credentials are saved. You can save the credentials using the cmdkey as [previously described](#).

If desired, you can dismount the Azure file share using the following PowerShell cmdlet.

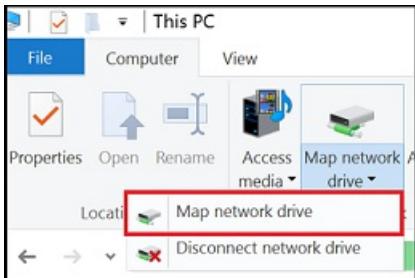
```
Remove-PSDrive -Name <desired-drive-letter>
```

Mount the Azure file share with File Explorer

NOTE

Note that the following instructions are shown on Windows 10 and may differ slightly on older releases.

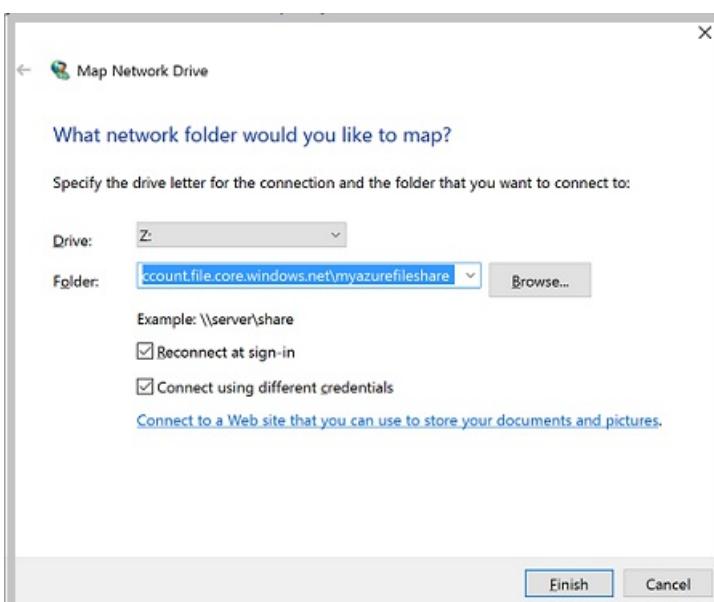
1. Open File Explorer. This can be done by opening from the Start Menu, or by pressing Win+E shortcut.
2. Navigate to the **This PC** item on the left-hand side of the window. This will change the menus available in the ribbon. Under the Computer menu, select **Map network drive**.



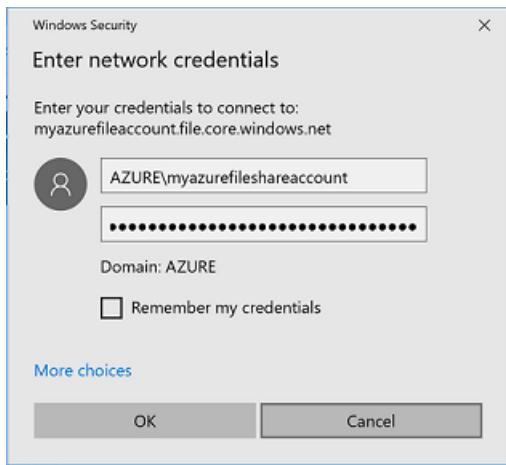
3. Copy the UNC path from the **Connect** pane in the Azure portal.

A screenshot of the Azure portal's 'Connect' pane for a file share named 'myazurefileshare'. The title is 'Connecting from Windows'. It instructs the user to run the command 'net use [drive letter] \\myazurefileaccount.file.core.windows.net\myazurefiles /u:AZURE\myazurefileaccount mehLWRwJkxSZTBFs8QFd7Xl3qjwF8Tojea2Eu4BfT0e4/aIobuB1upW'. The UNC path '\\myazurefileaccount.file.core.windows.net\myazurefiles' is highlighted with a red box.

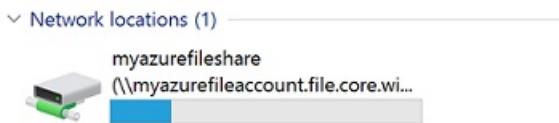
4. Select the drive letter and enter the UNC path.



5. Use the storage account name prepended with **AZURE** as the username and a storage account key as the password.



6. Use Azure file share as desired.



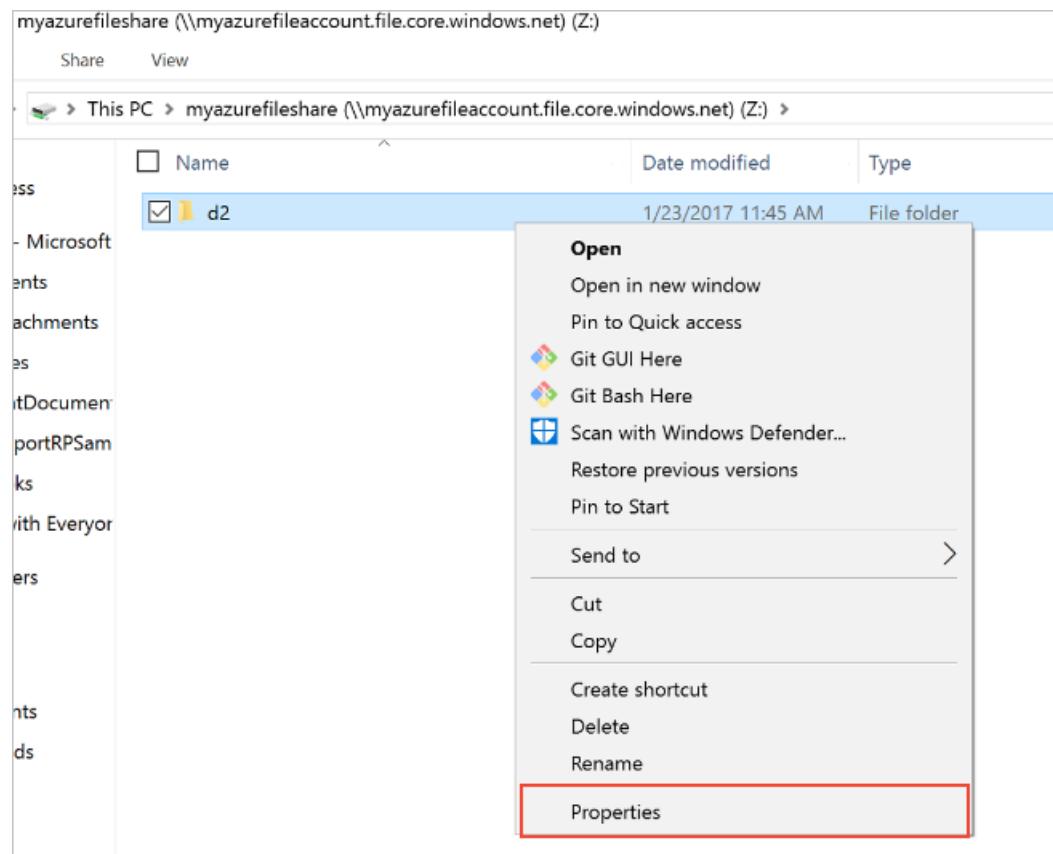
7. When you are ready to dismount the Azure file share, you can do so by right-clicking on the entry for the share under the **Network locations** in File Explorer and selecting **Disconnect**.

Accessing share snapshots from Windows

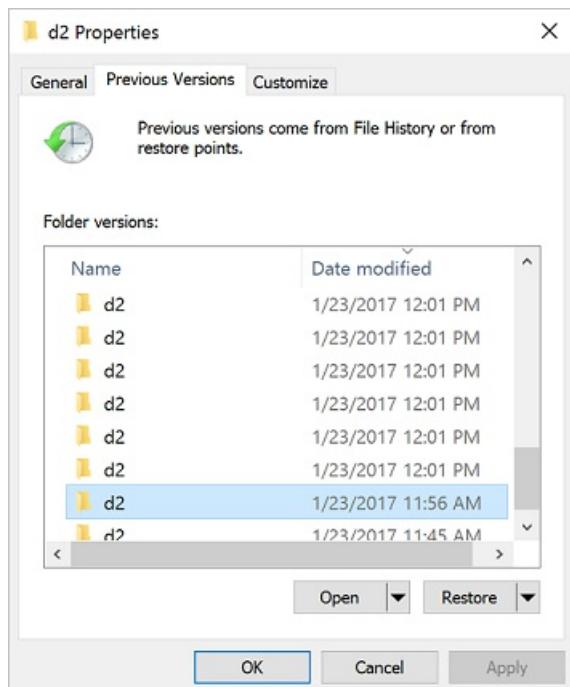
If you have taken a share snapshot, either manually or automatically through a script or service like Azure Backup, you can view previous versions of a share, a directory, or a particular file from file share on Windows. You can take a share snapshot from the [Azure portal](#), [Azure PowerShell](#), and [Azure CLI](#).

List previous versions

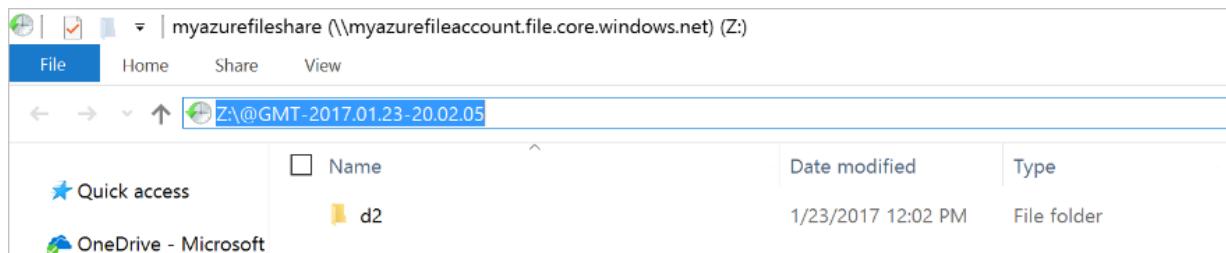
Browse to the item or parent item that needs to be restored. Double-click to go to the desired directory. Right-click and select **Properties** from the menu.



Select **Previous Versions** to see the list of share snapshots for this directory. The list might take a few seconds to load, depending on the network speed and the number of share snapshots in the directory.

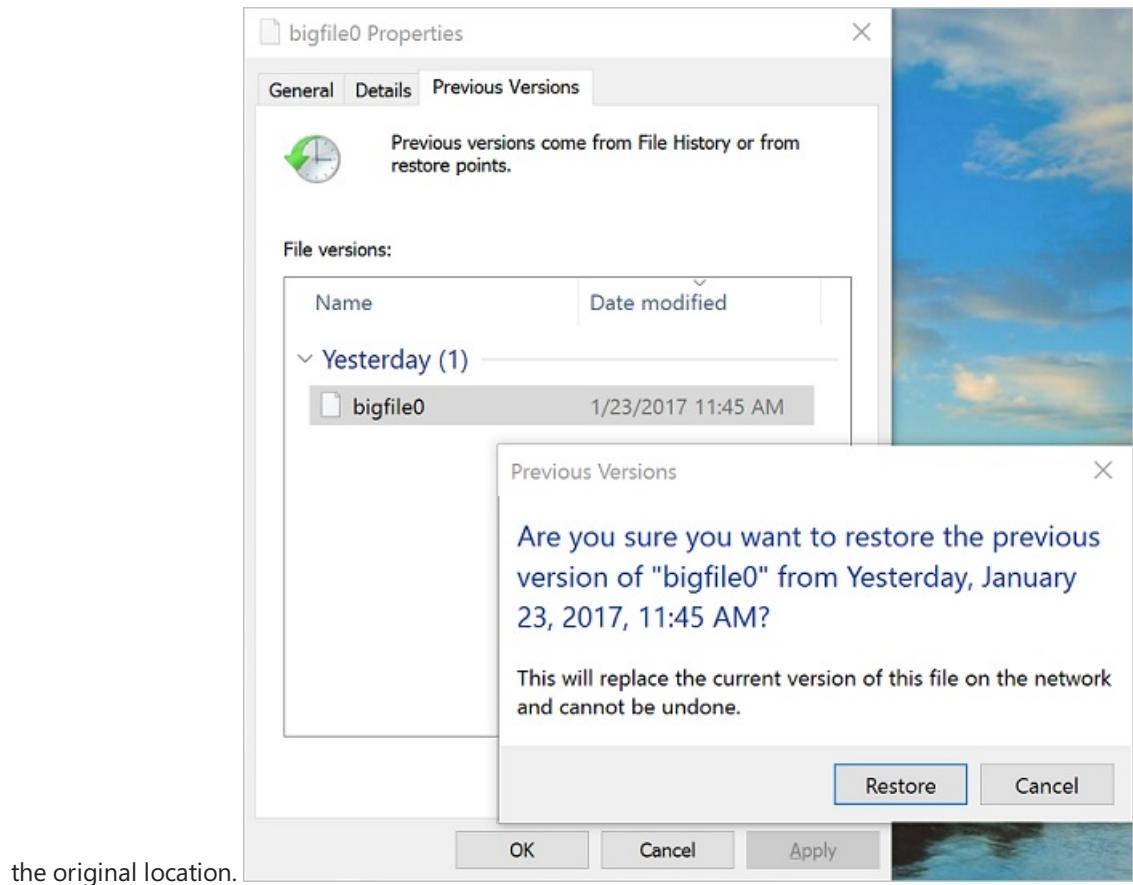


You can select **Open** to open a particular snapshot.



Restore from a previous version

Select **Restore** to copy the contents of the entire directory recursively at the share snapshot creation time to



the original location.

Securing Windows/Windows Server

In order to mount an Azure file share on Windows, port 445 must be accessible. Many organizations block port 445 because of the security risks inherent with SMB 1. SMB 1, also known as CIFS (Common Internet File System), is a legacy file system protocol included with Windows and Windows Server. SMB 1 is an outdated, inefficient, and most importantly insecure protocol. The good news is that Azure Files does not support SMB 1, and all supported versions of Windows and Windows Server make it possible to remove or disable SMB 1. We always [strongly recommend](#) removing or disabling the SMB 1 client and server in Windows before using Azure file shares in production.

The following table provides detailed information on the status of SMB 1 each version of Windows:

WINDOWS VERSION	SMB 1 DEFAULT STATUS	DISABLE/REMOVE METHOD
Windows Server 2019	Disabled	Remove with Windows feature
Windows Server, versions 1709+	Disabled	Remove with Windows feature
Windows 10, versions 1709+	Disabled	Remove with Windows feature
Windows Server 2016	Enabled	Remove with Windows feature
Windows 10, versions 1507, 1607, and 1703	Enabled	Remove with Windows feature
Windows Server 2012 R2	Enabled	Remove with Windows feature
Windows 8.1	Enabled	Remove with Windows feature
Windows Server 2012	Enabled	Disable with Registry

WINDOWS VERSION	SMB 1 DEFAULT STATUS	DISABLE/REMOVE METHOD
Windows Server 2008 R2	Enabled	Disable with Registry
Windows 7	Enabled	Disable with Registry

Auditing SMB 1 usage

Applies to Windows Server 2019, Windows Server semi-annual channel (versions 1709 and 1803), Windows Server 2016, Windows 10 (versions 1507, 1607, 1703, 1709, and 1803), Windows Server 2012 R2, and Windows 8.1

Before removing SMB 1 in your environment, you may wish to audit SMB 1 usage to see if any clients will be broken by the change. If any requests are made against SMB shares with SMB 1, an audit event will be logged in the event log under `Applications and Services Logs > Microsoft > Windows > SMBServer > Audit`.

NOTE

To enable auditing support on Windows Server 2012 R2 and Windows 8.1, install at least [KB4022720](#).

To enable auditing, execute the following cmdlet from an elevated PowerShell session:

```
Set-SmbServerConfiguration -AuditSmb1Access $true
```

Removing SMB 1 from Windows Server

Applies to Windows Server 2019, Windows Server semi-annual channel (versions 1709 and 1803), Windows Server 2016, Windows Server 2012 R2

To remove SMB 1 from a Windows Server instance, execute the following cmdlet from an elevated PowerShell session:

```
Remove-WindowsFeature -Name FS-SMB1
```

To complete the removal process, restart your server.

NOTE

Starting with Windows 10 and Windows Server version 1709, SMB 1 is not installed by default and has separate Windows features for the SMB 1 client and SMB 1 server. We always recommend leaving both the SMB 1 server (`FS-SMB1-SERVER`) and the SMB 1 client (`FS-SMB1-CLIENT`) uninstalled.

Removing SMB 1 from Windows client

Applies to Windows 10 (versions 1507, 1607, 1703, 1709, and 1803) and Windows 8.1

To remove SMB 1 from your Windows client, execute the following cmdlet from an elevated PowerShell session:

```
Disable-WindowsOptionalFeature -Online -FeatureName SMB1Protocol
```

To complete the removal process, restart your PC.

Disabling SMB 1 on legacy versions of Windows/Windows Server

Applies to Windows Server 2012, Windows Server 2008 R2, and Windows 7

SMB 1 cannot be completely removed on legacy versions of Windows/Windows Server, but it can be disabled through the Registry. To disable SMB 1, create a new registry key `SMB1` of type `DWORD` with a value of `0` under `HKEY_LOCAL_MACHINE > SYSTEM > CurrentControlSet > Services > LanmanServer > Parameters`.

You can easily accomplish this with the following PowerShell cmdlet as well:

```
Set-ItemProperty -Path "HKLM:\SYSTEM\CurrentControlSet\Services\LanmanServer\Parameters" SMB1 -Type DWORD  
-Value 0 -Force
```

After creating this registry key, you must restart your server to disable SMB 1.

SMB resources

- [Stop using SMB 1](#)
- [SMB 1 Product Clearinghouse](#)
- [Discover SMB 1 in your environment with DSCEA](#)
- [Disabling SMB 1 through Group Policy](#)

Next steps

See these links for more information about Azure Files:

- [Planning for an Azure Files deployment](#)
- [FAQ](#)
- [Troubleshooting on Windows](#)

Use Azure Files with Linux

1/8/2020 • 9 minutes to read • [Edit Online](#)

Azure Files is Microsoft's easy to use cloud file system. Azure file shares can be mounted in Linux distributions using the [SMB kernel client](#). This article shows two ways to mount an Azure file share: on-demand with the `mount` command and on-boot by creating an entry in `/etc/fstab`.

The recommended way to mount an Azure file share on Linux is using SMB 3.0. By default, Azure Files requires encryption in transit, which is only supported by SMB 3.0. Azure Files also supports SMB 2.1, which does not support encryption in transit, but you may not mount Azure file shares with SMB 2.1 from another Azure region or on-premises for security reasons. Unless your application specifically requires SMB 2.1, there is little reason to use it since most popular, recently released Linux distributions support SMB 3.0:

	SMB 2.1 (MOUNTS ON VMS WITHIN SAME AZURE REGION)	SMB 3.0 (MOUNTS FROM ON PREMISES AND CROSS-REGION)
Ubuntu	14.04+	16.04+
Red Hat Enterprise Linux (RHEL)	7+	7.5+
CentOS	7+	7.5+
Debian	8+	10+
openSUSE	13.2+	42.3+
SUSE Linux Enterprise Server	12+	12 SP3+

If you're using a Linux distribution not listed in the above table, you can check to see if your Linux distribution supports SMB 3.0 with encryption by checking the Linux kernel version. SMB 3.0 with encryption was added to Linux kernel version 4.11. The `uname` command will return the version of the Linux kernel in use:

```
uname -r
```

Prerequisites

- **Ensure the cifs-utils package is installed.**

The cifs-utils package can be installed using the package manager on the Linux distribution of your choice.

On **Ubuntu** and **Debian-based** distributions, use the `apt` package manager:

```
sudo apt update
sudo apt install cifs-utils
```

On **Fedor**a, **Red Hat Enterprise Linux 8+**, and **CentOS 8 +**, use the `dnf` package manager:

```
sudo dnf install cifs-utils
```

On older versions of **Red Hat Enterprise Linux** and **CentOS**, use the `yum` package manager:

```
sudo yum install cifs-utils
```

On **openSUSE**, use the `zypper` package manager:

```
sudo zypper install cifs-utils
```

On other distributions, use the appropriate package manager or [compile from source](#)

- **The most recent version of the Azure Command Line Interface (CLI).** For more information on how to install the Azure CLI, see [Install the Azure CLI](#) and select your operating system. If you prefer to use the Azure PowerShell module in PowerShell 6+, you may, however the instructions below are presented for the Azure CLI.
- **Ensure port 445 is open:** SMB communicates over TCP port 445 - check to see if your firewall is not blocking TCP ports 445 from client machine. Replace and

```
resourceGroupName=<your-resource-group>
storageAccountName=<your-storage-account>

# This command assumes you have logged in with az login
httpEndpoint=$(az storage account show \
    --resource-group $resourceGroupName \
    --name $storageAccountName \
    --query "primaryEndpoints.file" | tr -d "'")
smbPath=$(echo $httpEndpoint | cut -c7-$(expr length $httpEndpoint))
fileHost=$(echo $smbPath | tr -d "/")

nc -zvw3 $fileHost 445
```

If the connection was successful, you should see something similar to the following output:

```
Connection to <your-storage-account> 445 port [tcp/microsoft-ds] succeeded!
```

If you are unable to open up port 445 on your corporate network or are blocked from doing so by an ISP, you may use a VPN connection or ExpressRoute to work around port 445. For more information, see [Networking considerations for direct Azure file share access..](#)

Mounting Azure file share

To use an Azure file share with your Linux distribution, you must create a directory to serve as the mount point for the Azure file share. A mount point can be created anywhere on your Linux system, but it's common convention to create this under /mnt. After the mount point, you use the `mount` command to access the Azure file share.

You can mount the same Azure file share to multiple mount points if desired.

Mount the Azure file share on-demand with `mount`

1. **Create a folder for the mount point:** Replace `<your-resource-group>`, `<your-storage-account>`, and `<your-file-share>` with the appropriate information for your environment:

```

resourceGroupName=<your-resource-group>
storageAccountName=<your-storage-account>
fileShareName=<your-file-share>

mntPath="/mnt/$storageAccountName/$fileShareName"

sudo mkdir -p $mntPath

```

2. **Use the mount command to mount the Azure file share.** In the example below, the local Linux file and folder permissions default 0755, which means read, write, and execute for the owner (based on the file/directory Linux owner), read and execute for users in owner group, and read and execute for others on the system. You can use the `uid` and `gid` mount options to set the user ID and group ID for the mount. You can also use `dir_mode` and `file_mode` to set custom permissions as desired. For more information on how to set permissions, see [UNIX numeric notation](#) on Wikipedia.

```

httpEndpoint=$(az storage account show \
--resource-group $resourceGroupName \
--name $storageAccountName \
--query "primaryEndpoints.file" | tr -d "'")
smbPath=$(echo $httpEndpoint | cut -c7-$(expr length $httpEndpoint))$fileShareName

storageAccountKey=$(az storage account keys list \
--resource-group $resourceGroupName \
--account-name $storageAccountName \
--query "[0].value" | tr -d "'")

sudo mount -t cifs $smbPath $mntPath -o
vers=3.0,username=$storageAccountName,password=$storageAccountKey,serverino

```

NOTE

The above mount command mounts with SMB 3.0. If your Linux distribution does not support SMB 3.0 with encryption or if it only supports SMB 2.1, you may only mount from an Azure VM within the same region as the storage account. To mount your Azure file share on a Linux distribution that does not support SMB 3.0 with encryption, you will need to [disable encryption in transit for the storage account](#).

When you are done using the Azure file share, you may use `sudo umount $mntPath` to unmount the share.

Create a persistent mount point for the Azure file share with `/etc/fstab`

1. **Create a folder for the mount point:** A folder for a mount point can be created anywhere on the file system, but it's common convention to create this under /mnt. For example, the following command creates a new directory, replace `<your-resource-group>`, `<your-storage-account>`, and `<your-file-share>` with the appropriate information for your environment:

```

resourceGroupName=<your-resource-group>
storageAccountName=<your-storage-account>
fileShareName=<your-file-share>

mntPath="/mnt/$storageAccountName/$fileShareName"

sudo mkdir -p $mntPath

```

2. **Create a credential file to store the username (the storage account name) and password (the storage account key) for the file share.**

```

if [ ! -d "/etc/smbcredentials" ]; then
    sudo mkdir "/etc/smbcredentials"
fi

storageAccountKey=$(az storage account keys list \
    --resource-group $resourceGroupName \
    --account-name $storageAccountName \
    --query "[0].value" | tr -d '')

smbCredentialFile="/etc/smbcredentials/$storageAccountName.cred"
if [ ! -f $smbCredentialFile ]; then
    echo "username=$storageAccountName" | sudo tee $smbCredentialFile > /dev/null
    echo "password=$storageAccountKey" | sudo tee -a $smbCredentialFile > /dev/null
else
    echo "The credential file $smbCredentialFile already exists, and was not modified."
fi

```

3. Change permissions on the credential file so only root can read or modify the password file.

Since the storage account key is essentially a super-administrator password for the storage account, setting the permissions on the file such that only root can access is important so that lower privilege users cannot retrieve the storage account key.

```
sudo chmod 600 $smbCredentialFile
```

4. Use the following command to append the following line to `/etc/fstab`

In the example below, the local Linux file and folder permissions default 0755, which means read, write, and execute for the owner (based on the file/directory Linux owner), read and execute for users in owner group, and read and execute for others on the system. You can use the `uid` and `gid` mount options to set the user ID and group ID for the mount. You can also use `dir_mode` and `file_mode` to set custom permissions as desired. For more information on how to set permissions, see [UNIX numeric notation](#) on Wikipedia.

```

httpEndpoint=$(az storage account show \
    --resource-group $resourceGroupName \
    --name $storageAccountName \
    --query "primaryEndpoints.file" | tr -d '')

smbPath=$(echo $httpEndpoint | cut -c7-$(expr length $httpEndpoint))$fileShareName

if [ -z "$(grep $smbPath\ $mntPath /etc/fstab)" ]; then
    echo "$smbPath $mntPath cifs nobrl,vers=3.0,credentials=$smbCredentialFile,serverino" | sudo
    tee -a /etc/fstab > /dev/null
else
    echo "/etc/fstab was not modified to avoid conflicting entries as this Azure file share was
already present. You may want to double check /etc/fstab to ensure the configuration is as desired."
fi

sudo mount -a

```

NOTE

The above mount command mounts with SMB 3.0. If your Linux distribution does not support SMB 3.0 with encryption or if it only supports SMB 2.1, you may only mount from an Azure VM within the same region as the storage account. To mount your Azure file share on a Linux distribution that does not support SMB 3.0 with encryption, you will need to [disable encryption in transit for the storage account](#).

Securing Linux

In order to mount an Azure file share on Linux, port 445 must be accessible. Many organizations block port

445 because of the security risks inherent with SMB 1. SMB 1, also known as CIFS (Common Internet File System), is a legacy file system protocol included with many Linux distributions. SMB 1 is an outdated, inefficient, and most importantly insecure protocol. The good news is that Azure Files does not support SMB 1, and starting with Linux kernel version 4.18, Linux makes it possible to disable SMB 1. We always **strongly recommend** disabling the SMB 1 on your Linux clients before using SMB file shares in production.

Starting with Linux kernel 4.18, the SMB kernel module, called `cifs` for legacy reasons, exposes a new module parameter (often referred to as *parm* by various external documentation), called `disable_legacy_dialects`. Although introduced in Linux kernel 4.18, some vendors have backported this change to older kernels that they support. For convenience, the following table details the availability of this module parameter on common Linux distributions.

DISTRIBUTION	CAN DISABLE SMB 1
Ubuntu 14.04-16.04	No
Ubuntu 18.04	Yes
Ubuntu 19.04+	Yes
Debian 8-9	No
Debian 10+	Yes
Fedora 29+	Yes
CentOS 7	No
CentOS 8+	Yes
Red Hat Enterprise Linux 6.x-7.x	No
Red Hat Enterprise Linux 8+	Yes
openSUSE Leap 15.0	No
openSUSE Leap 15.1+	Yes
openSUSE Tumbleweed	Yes
SUSE Linux Enterprise 11.x-12.x	No
SUSE Linux Enterprise 15	No
SUSE Linux Enterprise 15.1	No

You can check to see if your Linux distribution supports the `disable_legacy_dialects` module parameter via the following command.

```
sudo modinfo -p cifs | grep disable_legacy_dialects
```

This command should output the following message:

```
disable_legacy_dialects: To improve security it may be helpful to restrict the ability to override the default dialects (SMB2.1, SMB3 and SMB3.02) on mount with old dialects (CIFS/SMB1 and SMB2) since vers=1.0 (CIFS/SMB1) and vers=2.0 are weaker and less secure. Default: n/N/0 (bool)
```

Before disabling SMB 1, you must check to make sure that the SMB module is not currently loaded on your system (this happens automatically if you have mounted an SMB share). You can do this with the following command, which should output nothing if SMB is not loaded:

```
lsmod | grep cifs
```

To unload the module, first unmount all SMB shares (using the `umount` command as described above). You can identify all the mounted SMB shares on your system with the following command:

```
mount | grep cifs
```

Once you have unmounted all SMB file shares, it's safe to unload the module. You can do this with the `modprobe` command:

```
sudo modprobe -r cifs
```

You can manually load the module with SMB 1 unloaded using the `modprobe` command:

```
sudo modprobe cifs disable_legacy_dialects=Y
```

Finally, you can check the SMB module has been loaded with the parameter by looking at the loaded parameters in `/sys/module/cifs/parameters`:

```
cat /sys/module/cifs/parameters/disable_legacy_dialects
```

To persistently disable SMB 1 on Ubuntu and Debian-based distributions, you must create a new file (if you don't already have custom options for other modules) called `/etc/modprobe.d/local.conf` with the setting. You can do this with the following command:

```
echo "options cifs disable_legacy_dialects=Y" | sudo tee -a /etc/modprobe.d/local.conf > /dev/null
```

You can verify that this has worked by loading the SMB module:

```
sudo modprobe cifs  
cat /sys/module/cifs/parameters/disable_legacy_dialects
```

Feedback

Linux users, we want to hear from you!

The Azure Files for Linux users' group provides a forum for you to share feedback as you evaluate and adopt File storage on Linux. Email [Azure Files Linux Users](#) to join the users' group.

Next steps

See these links for more information about Azure Files:

- [Planning for an Azure Files deployment](#)
- [FAQ](#)
- [Troubleshooting](#)

Mount Azure file share over SMB with macOS

12/6/2019 • 2 minutes to read • [Edit Online](#)

[Azure Files](#) is Microsoft's easy-to-use cloud file system. Azure file shares can be mounted with the industry standard SMB 3 protocol by macOS El Capitan 10.11+. This article shows two different ways to mount an Azure file share on macOS: with the Finder UI and using the Terminal.

NOTE

Before mounting an Azure file share over SMB, we recommend disabling SMB packet signing. Not doing so may yield poor performance when accessing the Azure file share from macOS. Your SMB connection will be encrypted, so this does not affect the security of your connection. From the terminal, the following commands will disable SMB packet signing, as described by this [Apple support article on disabling SMB packet signing](#):

```
sudo -s
echo "[default]" >> /etc/nsmb.conf
echo "signing_required=no" >> /etc/nsmb.conf
exit
```

Prerequisites for mounting an Azure file share on macOS

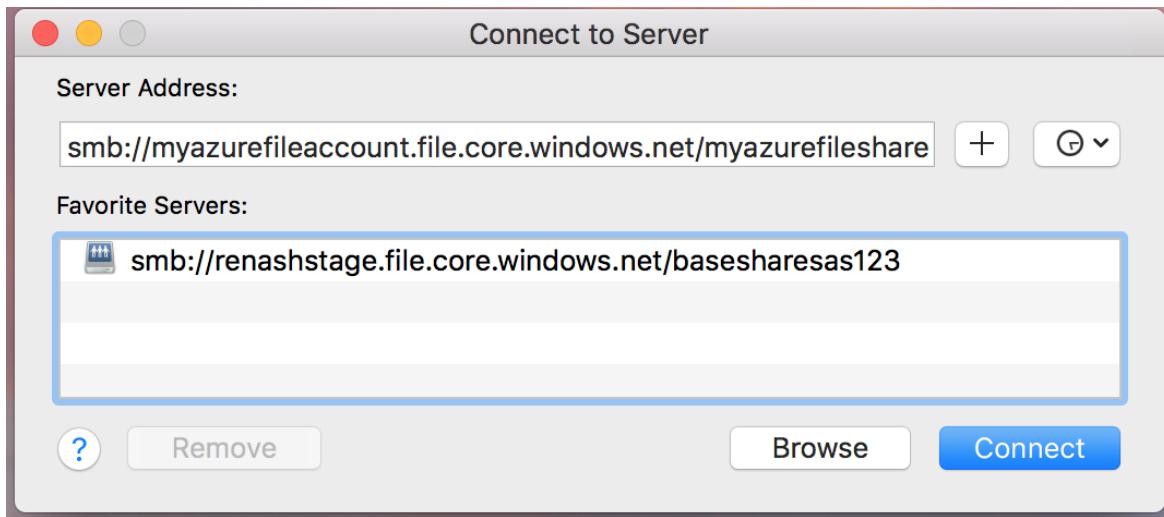
- **Storage account name:** To mount an Azure file share, you will need the name of the storage account.
- **Storage account key:** To mount an Azure file share, you will need the primary (or secondary) storage key. SAS keys are not currently supported for mounting.
- **Ensure port 445 is open:** SMB communicates over TCP port 445. On your client machine (the Mac), check to make sure your firewall is not blocking TCP port 445.

Mount an Azure file share via Finder

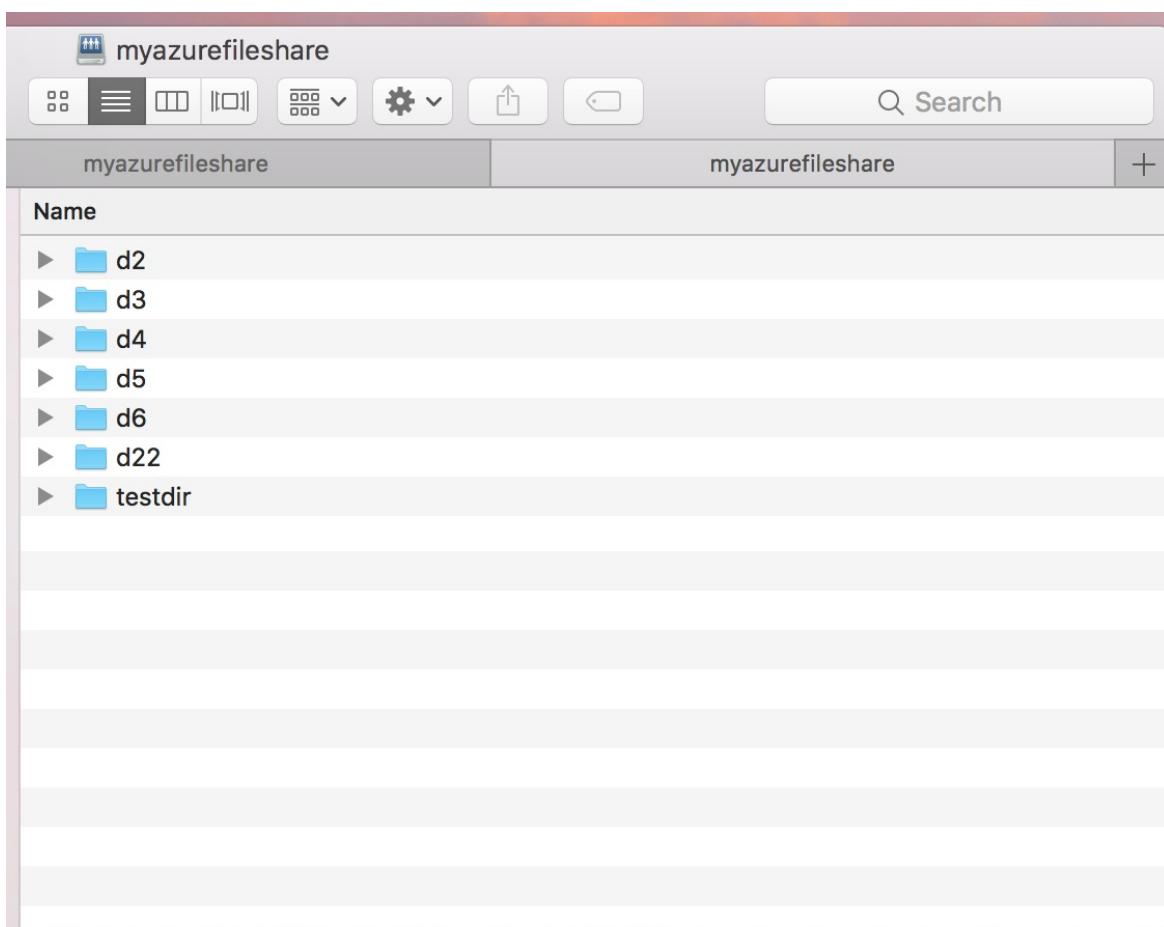
1. **Open Finder:** Finder is open on macOS by default, but you can ensure it is the currently selected application by clicking the "macOS face icon" on the dock:



2. **Select "Connect to Server" from the "Go" Menu:** Using the UNC path from the prerequisites, convert the beginning double backslash (\\\) to smb:// and all other backslashes (\) to forwards slashes (/). Your link should look like the following:



3. **Use the storage account name and storage account key when prompted for a username and password:** When you click "Connect" on the "Connect to Server" dialog, you will be prompted for the username and password (This will be autopopulated with your macOS username). You have the option of placing the storage account name/storage account key in your macOS Keychain.
4. **Use the Azure file share as desired:** After substituting the share name and storage account key in for the username and password, the share will be mounted. You may use this as you would normally use a local folder/file share, including dragging and dropping files into the file share:

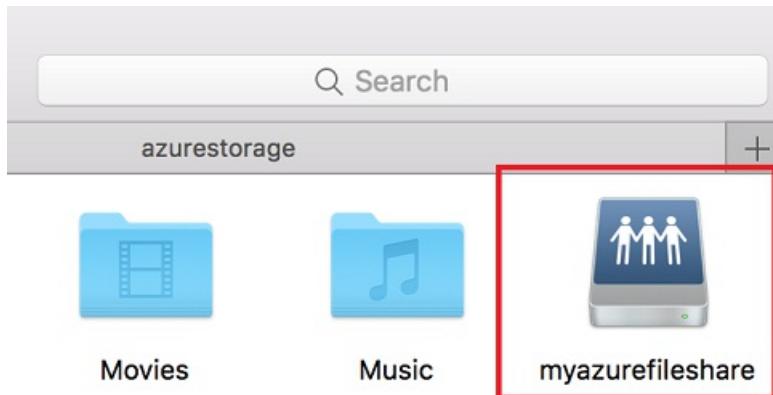


Mount an Azure file share via Terminal

1. Replace <storage-account-name> with the name of your storage account. Provide Storage Account Key as password when prompted.

```
mount_smbfs //<storage-account-name>@<storage-account-name>.file.core.windows.net/<share-name>
<desired-mount-point>
```

2. **Use the Azure file share as desired:** The Azure file share will be mounted at the mount point specified by the previous command.



Next steps

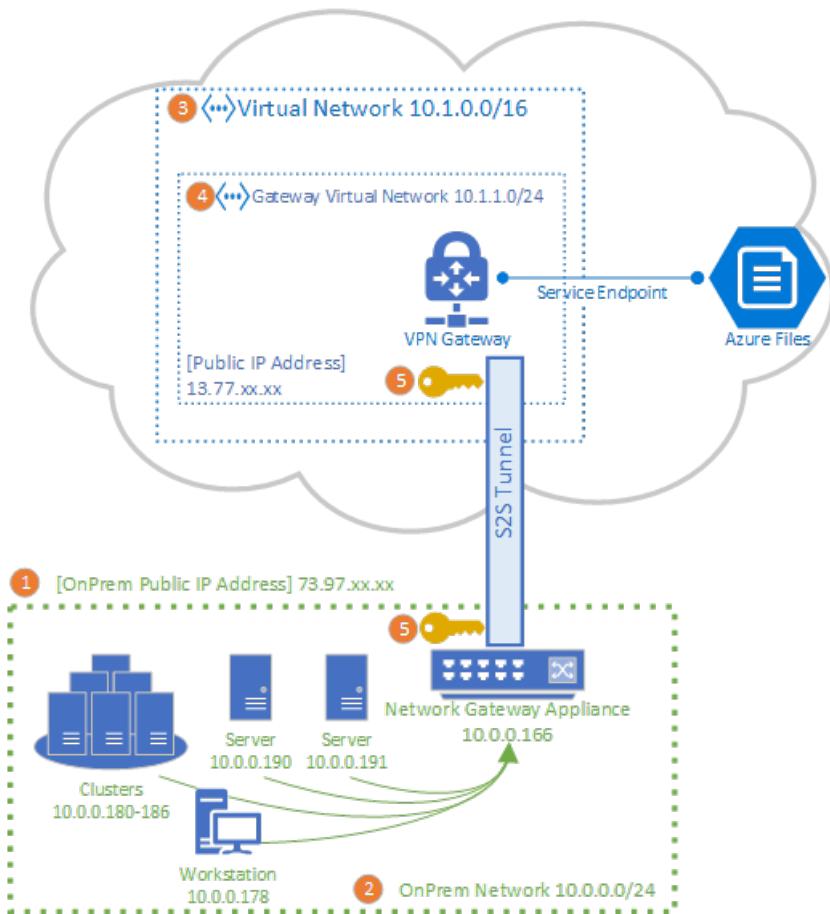
See these links for more information about Azure Files.

- [Apple Support Article - How to connect with File Sharing on your Mac](#)
- [FAQ](#)
- [Troubleshooting on Windows](#)
- [Troubleshooting on Linux](#)

Configure a Site-to-Site VPN for use with Azure Files

11/25/2019 • 9 minutes to read • [Edit Online](#)

You can use a Site-to-Site (S2S) VPN connection to mount your Azure file shares over SMB from your on-premises network, without opening up port 445. You can set up a Site-to-Site VPN using [Azure VPN Gateway](#), which is an Azure resource offering VPN services, and is deployed in a resource group alongside storage accounts or other Azure resources.



We strongly recommend that you read [Azure Files networking overview](#) before continuing with this how to article for a complete discussion of the networking options available for Azure Files.

The article details the steps to configure a Site-to-Site VPN to mount Azure file shares directly on-premises. If you're looking to route sync traffic for Azure File Sync over a Site-to-Site VPN, please see [configuring Azure File Sync proxy and firewall settings](#).

Prerequisites

- An Azure file share you would like to mount on-premises. You may use either a [standard](#) or a [premium Azure file share](#) with your Site-to-Site VPN.
- A network appliance or server in your on-premises datacenter that is compatible with Azure VPN Gateway. Azure Files is agnostic of the on-premises network appliance chosen but Azure VPN Gateway maintains a [list of tested devices](#). Different network appliances offer different features, performance characteristics, and management functionalities, so consider these when selecting a network appliance.

If you do not have an existing network appliance, Windows Server contains a built-in Server Role, Routing and Remote Access (RRAS), which may be used as the on-premises network appliance. To learn more

about how to configure Routing and Remote Access in Windows Server, see [RAS Gateway](#).

Add storage account to VNet

In the Azure portal, navigate to the storage account containing the Azure file share you would like to mount on-premises. In the table of contents for the storage account, select the **Firewalls and virtual networks** entry. Unless you added a virtual network to your storage account when you created it, the resulting pane should have the **Allow access from** radio button for **All networks** selected.

To add your storage account to the desired virtual network, select **Selected networks**. Under the **Virtual networks** subheading, click either **+ Add existing virtual network** or **+Add new virtual network** depending on the desired state. Creating a new virtual network will result in a new Azure resource being created. The new or existing VNet resource does not need to be in the same resource group or subscription as the storage account, however it must be in the same region as the storage account and the resource group and subscription you deploy your VNet into must match the one you will deploy your VPN Gateway into.

Virtual networks		
VIRTUAL NETWORK	SUBNET	ADDRESS RANGE
No network selected.		

If you add existing virtual network, you will be asked to select one or more subnets of that virtual network which the storage account should be added to. If you select a new virtual network, you will create a subnet as part of the creation of the virtual network, and you can add more later through the resulting Azure resource for the virtual network.

If you have not added a storage account to your subscription before, the Microsoft.Storage service endpoint will need to be added to the virtual network. This may take some time, and until this operation has completed, you will not be able to access the Azure file shares within that storage account, including via the VPN connection.

Deploy an Azure VPN Gateway

In the table of contents for the Azure portal, select **Create a new resource** and search for *Virtual network gateway*. Your virtual network gateway must be in the same subscription, Azure region, and resource group as the virtual network you deployed in the previous step (note that resource group is automatically selected when the virtual network is picked).

For the purposes of deploying an Azure VPN Gateway, you must populate the following fields:

- **Name:** The name of the Azure resource for the VPN Gateway. This name may be any name you find useful for your management.
- **Region:** The region into which the VPN Gateway will be deployed.
- **Gateway type:** For the purpose of deploying a Site-to-Site VPN, you must select **VPN**.
- **VPN type:** You may choose either *Route-based** or **Policy-based** depending on your VPN device. Route-based VPNs support IKEv2, while policy-based VPNs only support IKEv1. To learn more about the two types of VPN gateways, see [About policy-based and route-based VPN gateways](#)
- **SKU:** The SKU controls the number of allowed Site-to-Site tunnels and desired performance of the VPN. To select the appropriate SKU for your use case, consult the [Gateway SKU](#) listing. The SKU of the VPN Gateway may be changed later if necessary.
- **Virtual network:** The virtual network you created in the previous step.
- **Public IP address:** The IP address of VPN Gateway that will be exposed to the internet. Likely, you will need to create a new IP address, however you may also use an existing unused IP address if that is appropriate. If

you select to **Create new**, a new IP address Azure resource will be created in the same resource group as the VPN Gateway and the **Public IP address name** will be the name of the newly created IP address. If you select **Use existing**, you must select the existing unused IP address.

- **Enable active-active mode:** Only select **Enabled** if you are creating an active-active gateway configuration, otherwise leave **Disabled** selected. To learn more about active-active mode, see [Highly available cross-premises and VNet-to-VNet connectivity](#).
- **Configure BGP ASN:** Only select **Enabled** if your configuration specifically requires this setting. To learn more about this setting, see [About BGP with Azure VPN Gateway](#).

Select **Review + create** to create the VPN Gateway. A VPN Gateway may take up to 45 minutes to fully create and deploy.

Create a local network gateway for your on-premises gateway

A local network gateway is an Azure resource that represents your on-premises network appliance. In the table of contents for the Azure portal, select **Create a new resource** and search for *local network gateway*. The local network gateway is an Azure resource that will be deployed alongside your storage account, virtual network, and VPN Gateway, but does not need to be in the same resource group or subscription as the storage account.

For the purposes of deploying the local network gateway resource, you must populate the following fields:

- **Name:** The name of the Azure resource for the local network gateway. This name may be any name you find useful for your management.
- **IP address:** The public IP address of your local gateway on-premises.
- **Address space:** The address ranges for the network this local network gateway represents. You can add multiple address space ranges, but make sure that the ranges you specify here do not overlap with ranges of other networks that you want to connect to.
- **Configure BGP settings:** Only configure BGP settings if your configuration requires this setting. To learn more about this setting, see [About BGP with Azure VPN Gateway](#).
- **Subscription:** The desired subscription. This does not need to match the subscription used for the VPN Gateway or the storage account.
- **Resource group:** The desired resource group. This does not need to match the resource group used for the VPN Gateway or the storage account.
- **Location:** The Azure Region the local network gateway resource should be created in. This should match the region you selected for the VPN Gateway and the storage account.

Select **Create** to create the local network gateway resource.

Configure on-premises network appliance

The specific steps to configure your on-premises network appliance depend based on the network appliance your organization has selected. Depending on the device your organization has chosen, the [list of tested devices](#) may have a link out to your device vendor's instructions for configuring with Azure VPN Gateway.

Create private endpoint (preview)

Creating a private endpoint for your storage account gives your storage account an IP address within the IP address space of your virtual network. When you mount your Azure file share from on-premises using this private IP address, the routing rules autodefined by the VPN installation will route your mount request to the storage account via the VPN.

In the storage account blade, select **Private endpoint connections** in the left-hand table of contents and + **Private endpoint** to create a new private endpoint. The resulting wizard has multiple pages to complete:

Create a private endpoint (Preview)

1 Basics 2 Resource 3 Configuration 4 Tags 5 Review + create

Use private endpoints to privately connect to a service or resource. Your private endpoint must be in the same region as your virtual network, but can be in a different region from the private link resource that you are connecting to. [Learn more](#)

Project details

Subscription * ⓘ

Resource group * ⓘ [Create new](#)

Instance details

Name *

Region *

On the **Basics** tab, select the desired resource group, name, and region for your private endpoint. These can be whatever you want, they don't have to match the storage account in anyway, although you must create the private endpoint in the same region as the virtual network you wish to create the private endpoint in.

On the **Resource** tab, select the radio button for **Connect to an Azure resource in my directory**. Under **Resource type**, select **Microsoft.Storage/storageAccounts** for the resource type. The **Resource** field is the storage account with the Azure file share you wish to connect to. Target sub-resource is **file**, since this is for Azure Files.

The **Configuration** tab allows you to select the specific virtual network and subnet you would like to add your private endpoint to. Select the virtual network you created above. You must select a distinct subnet from the subnet you added your service endpoint to above.

The **Configuration** tab also allows you to set up a private DNS zone. This is not required, but allows you to use a friendly UNC path (such as `\mystorageaccount.privatelink.file.core.windows.net\myshare`) instead of a UNC path with an IP address to mount the Azure file share. This may also be done with your own DNS servers within your virtual network.

Click **Review + create** to create the private endpoint. Once the private endpoint has been created, you will see two new resources: a private endpoint resource and a paired virtual network interface. The virtual network interface resource will have the dedicated private IP of the storage account.

Create the Site-to-Site connection

To complete the deployment of a S2S VPN, you must create a connection between your on-premises network appliance (represented by the local network gateway resource) and the VPN Gateway. To do this, navigate to the VPN Gateway you created above. In the table of contents for the VPN Gateway, select **Connections**, and click **Add**. The resulting **Add connection** pane requires the following fields:

- **Name:** The name of the connection. A VPN Gateway can host multiple connections, so pick a name helpful for your management that will distinguish this particular connection.
- **Connection type:** Since this a S2S connection, select **Site-to-site (IPSec)** in the drop-down list.
- **Virtual network gateway:** This field is auto-selected to the VPN Gateway you're making the connection to and can't be changed.
- **Local network gateway:** This is the local network gateway you want to connect to your VPN Gateway. The resulting selection pane should have the name of the local network gateway you created above.

- **Shared key (PSK)**: A mixture of letters and numbers, used to establish encryption for the connection. The same shared key must be used in both the virtual network and local network gateways. If your gateway device doesn't provide one, you can make one up here and provide it to your device.

Select **OK** to create the connection. You can verify the connection has been made successfully through the **Connections** page.

Mount Azure file share

The final step in configuring a S2S VPN is verifying that it works for Azure Files. You can do this by mounting your Azure file share on-premises with your preferred OS. See the instructions to mount by OS here:

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

See also

- [Azure Files networking overview](#)
- [Configure a Point-to-Site \(P2S\) VPN on Windows for use with Azure Files](#)
- [Configure a Point-to-Site \(P2S\) VPN on Linux for use with Azure Files](#)

Configure a Point-to-Site (P2S) VPN on Windows for use with Azure Files

12/31/2019 • 10 minutes to read • [Edit Online](#)

You can use a Point-to-Site (P2S) VPN connection to mount your Azure file shares over SMB from outside of Azure, without opening up port 445. A Point-to-Site VPN connection is a VPN connection between Azure and an individual client. To use a P2S VPN connection with Azure Files, a P2S VPN connection will need to be configured for each client that wants to connect. If you have many clients that need to connect to your Azure file shares from your on-premises network, you can use a Site-to-Site (S2S) VPN connection instead of a Point-to-Site connection for each client. To learn more, see [Configure a Site-to-Site VPN for use with Azure Files](#).

We strongly recommend that you read [Networking considerations for direct Azure file share access](#) before continuing with this how to article for a complete discussion of the networking options available for Azure Files.

The article details the steps to configure a Point-to-Site VPN on Windows (Windows client and Windows Server) to mount Azure file shares directly on-premises. If you're looking to route Azure File Sync traffic over a VPN, please see [configuring Azure File Sync proxy and firewall settings](#).

Prerequisites

- The most recent version of the Azure PowerShell module. For more information on how to install the Azure PowerShell, see [Install the Azure PowerShell module](#) and select your operating system. If you prefer to use the Azure CLI on Windows, you may, however the instructions below are presented for Azure PowerShell.
- The Azure Private DNS PowerShell module. This is not currently distributed as part of the Azure PowerShell module, so this may be installed with the following method:

```
if ($PSVersionTable.PSVersion -ge [System.Version]::new(6, 0)) {
    Install-Module -Name Az.PrivateDns -AllowClobber -AllowPrerelease
} else {
    Install-Module -Name Az.PrivateDns -RequiredVersion "0.1.3"
}

Import-Module -Name Az.PrivateDns
```

- An Azure file share you would like to mount on-premises. You may use either a [standard](#) or a [premium Azure file share](#) with your Point-to-Site VPN.

Deploy a virtual network

To access your Azure file share and other Azure resources from on-premises via a Point-to-Site VPN, you must create a virtual network, or VNet. The P2S VPN connection you will automatically create is a bridge between your on-premises Windows machine and this Azure virtual network.

The following PowerShell will create an Azure virtual network with three subnets: one for your storage account's service endpoint, one for your storage account's private endpoint, which is required to access the storage account on-premises without creating custom routing for the public IP of the storage account that may change, and one for your virtual network gateway that provides the VPN service.

Remember to replace `<region>`, `<resource-group>`, and `<desired-vnet-name>` with the appropriate values for your environment.

```

$region = "<region>"
$resourceGroupName = "<resource-group>"
$virtualNetworkName = "<desired-vnet-name>

$virtualNetwork = New-AzVirtualNetwork ` 
    -ResourceGroupName $resourceGroupName ` 
    -Name $virtualNetworkName ` 
    -Location $region ` 
    -AddressPrefix "192.168.0.0/16"

Add-AzVirtualNetworkSubnetConfig ` 
    -Name "ServiceEndpointSubnet" ` 
    -AddressPrefix "192.168.0.0/24" ` 
    -VirtualNetwork $virtualNetwork ` 
    -ServiceEndpoint "Microsoft.Storage" ` 
    -WarningAction SilentlyContinue | Out-Null

Add-AzVirtualNetworkSubnetConfig ` 
    -Name "PrivateEndpointSubnet" ` 
    -AddressPrefix "192.168.1.0/24" ` 
    -VirtualNetwork $virtualNetwork ` 
    -WarningAction SilentlyContinue | Out-Null

Add-AzVirtualNetworkSubnetConfig ` 
    -Name "GatewaySubnet" ` 
    -AddressPrefix "192.168.2.0/24" ` 
    -VirtualNetwork $virtualNetwork ` 
    -WarningAction SilentlyContinue | Out-Null

$virtualNetwork | Set-AzVirtualNetwork | Out-Null
$virtualNetwork = Get-AzVirtualNetwork ` 
    -ResourceGroupName $resourceGroupName ` 
    -Name $virtualNetworkName

$serviceEndpointSubnet = $virtualNetwork.Subnets | ` 
    Where-Object { $_.Name -eq "ServiceEndpointSubnet" } 
$privateEndpointSubnet = $virtualNetwork.Subnets | ` 
    Where-Object { $_.Name -eq "PrivateEndpointSubnet" } 
$gatewaySubnet = $virtualNetwork.Subnets | ` 
    Where-Object { $_.Name -eq "GatewaySubnet" }

```

Restrict the storage account to the virtual network

By default when you create a storage account, you can access it from anywhere in the world as long as you have the means to authenticate your request (such as with your Active Directory identity or with the storage account key). To restrict access to this storage account to the virtual network you just created, you need to create a network rule set that allows access within the virtual network and denies all other access.

Restricting the storage account to the virtual network requires the use of a service endpoint. The service endpoint is a networking construct by which the public DNS/public IP can be accessed only from within the virtual network. Since the public IP address is not guaranteed to remain the same, we ultimately want to use a private endpoint rather than a service endpoint for the storage account, however it is not possible to restrict the storage account unless a service endpoint is also exposed.

Remember to replace <storage-account-name> with the storage account you want to access.

```
$storageAccountName = "<storage-account-name>"  
  
$storageAccount = Get-AzStorageAccount `  
    -ResourceGroupName $resourceGroupName `  
    -Name $storageAccountName  
  
$networkRule = Add-AzStorageAccountNetworkRule `  
    -ResourceGroupName $resourceGroupName `  
    -Name $storageAccountName `  
    -VirtualNetworkResourceId $serviceEndpointSubnet.Id  
  
Update-AzStorageAccountNetworkRuleSet `  
    -ResourceGroupName $resourceGroupName `  
    -Name $storageAccountName `  
    -Bypass AzureServices `  
    -DefaultAction Deny `  
    -VirtualNetworkRule $networkRule | Out-Null
```

Create a private endpoint (preview)

Creating a private endpoint for your storage account gives your storage account an IP address within the IP address space of your virtual network. When you mount your Azure file share from on-premises using this private IP address, the routing rules autodefined by the VPN installation will route your mount request to the storage account via the VPN.

```

$internalVnet = Get-AzResource ` 
    -ResourceId $virtualNetwork.Id ` 
    -ApiVersion "2019-04-01"

$internalVnet.Properties.subnets[1].properties.privateEndpointNetworkPolicies = "Disabled"
$internalVnet | Set-AzResource -Force | Out-Null

$privateEndpointConnection = New-AzPrivateLinkServiceConnection ` 
    -Name "myConnection" ` 
    -PrivateLinkServiceId $storageAccount.Id ` 
    -GroupId "file"

$privateEndpoint = New-AzPrivateEndpoint ` 
    -ResourceGroupName $resourceGroupName ` 
    -Name "$storageAccountName-privateEndpoint" ` 
    -Location $region ` 
    -Subnet $privateEndpointSubnet ` 
    -PrivateLinkServiceConnection $privateEndpointConnection

$zone = Get-AzPrivateDnsZone -ResourceGroupName $resourceGroupName
if ($null -eq $zone) {
    $zone = New-AzPrivateDnsZone ` 
        -ResourceGroupName $resourceGroupName ` 
        -Name "privatelink.file.core.windows.net"
} else {
    $zone = $zone[0]
}

$link = New-AzPrivateDnsVirtualNetworkLink ` 
    -ResourceGroupName $resourceGroupName ` 
    -ZoneName $zone.Name ` 
    -Name ($virtualNetwork.Name + "-link") ` 
    -VirtualNetworkId $virtualNetwork.Id

$internalNic = Get-AzResource ` 
    -ResourceId $privateEndpoint.NetworkInterfaces[0].Id ` 
    -ApiVersion "2019-04-01"

foreach($ipconfig in $internalNic.Properties.ipConfigurations) {
    foreach($fqdn in $ipconfig.properties.privateLinkConnectionProperties.fqdns) {
        $recordName = $fqdn.split('.', 2)[0]
        $dnsZone = $fqdn.split('.', 2)[1]
        New-AzPrivateDnsRecordSet ` 
            -ResourceGroupName $resourceGroupName ` 
            -Name $recordName ` 
            -RecordType A ` 
            -ZoneName $zone.Name ` 
            -Ttl 600 ` 
            -PrivateDnsRecords (New-AzPrivateDnsRecordConfig ` 
                -IPv4Address $ipconfig.properties.privateIPAddress) | Out-Null
    }
}
}

```

Create root certificate for VPN authentication

In order for VPN connections from your on-premises Windows machines to be authenticated to access your virtual network, you must create two certificates: a root certificate, which will be provided to the virtual machine gateway, and a client certificate, which will be signed with the root certificate. The following PowerShell creates the root certificate; the client certificate will be created after the Azure virtual network gateway is created with information from the gateway.

```

$rootcertname = "CN=P2SRootCert"
$certLocation = "Cert:\CurrentUser\My"
$vpnTemp = "C:\vpn-temp\
$exportedencodedrootcertpath = $vpnTemp + "P2SRootCertencoded.cer"
$exportedrootcertpath = $vpnTemp + "P2SRootCert.cer"

if (-Not (Test-Path $vpnTemp)) {
    New-Item -ItemType Directory -Force -Path $vpnTemp | Out-Null
}

if ($PSVersionTable.PSVersion -ge [System.Version]::new(6, 0)) {
    Install-Module WindowsCompatibility
    Import-WinModule PKI
}

$rootcert = New-SelfSignedCertificate `

    -Type Custom `

    -KeySpec Signature `

    -Subject $rootcertname `

    -KeyExportPolicy Exportable `

    -HashAlgorithm sha256 `

    -KeyLength 2048 `

    -CertStoreLocation $certLocation `

    -KeyUsageProperty Sign `

    -KeyUsage CertSign

Export-Certificate `

    -Cert $rootcert `

    -FilePath $exportedencodedrootcertpath `

    -NoClobber | Out-Null

certutil -encode $exportedencodedrootcertpath $exportedrootcertpath | Out-Null

$rawRootCertificate = Get-Content -Path $exportedrootcertpath

[System.String]$rootCertificate = ""
foreach($line in $rawRootCertificate) {
    if ($line -notlike "*Certificate*") {
        $rootCertificate += $line
    }
}

```

Deploy virtual network gateway

The Azure virtual network gateway is the service that your on-premises Windows machines will connect to. Deploying this service requires two basic components: a public IP that will identify the gateway to your clients wherever they are in the world and a root certificate you created earlier which will be used to authenticate your clients.

Remember to replace <desired-vpn-name-here> with the name you would like for these resources.

NOTE

Deploying the Azure virtual network gateway can take up to 45 minutes. While this resource is being deployed, this PowerShell script will block for the deployment to be completed. This is expected.

```

$vpnName = "<desired-vpn-name-here>"
$publicIpAddressName = "$vpnName-PublicIP"

$publicIPAddress = New-AzPublicIpAddress ` 
    -ResourceGroupName $resourceGroupName ` 
    -Name $publicIpAddressName ` 
    -Location $region ` 
    -Sku Basic ` 
    -AllocationMethod Dynamic

$gatewayIpConfig = New-AzVirtualNetworkGatewayIpConfig ` 
    -Name "vnetGatewayConfig" ` 
    -SubnetId $gatewaySubnet.Id ` 
    -PublicIpAddressId $publicIPAddress.Id

$azRootCertificate = New-AzVpnClientRootCertificate ` 
    -Name "P2SRootCert" ` 
    -PublicCertData $rootCertificate

$vpn = New-AzVirtualNetworkGateway ` 
    -ResourceGroupName $resourceGroupName ` 
    -Name $vpnName ` 
    -Location $region ` 
    -GatewaySku VpnGw1 ` 
    -GatewayType Vpn ` 
    -VpnType RouteBased ` 
    -IpConfigurations $gatewayIpConfig ` 
    -VpnClientAddressPool "172.16.201.0/24" ` 
    -VpnClientProtocol IkeV2 ` 
    -VpnClientRootCertificates $azRootCertificate

```

Create client certificate

The client certificate is created with the URI of the virtual network gateway. This certificate is signed with the root certificate you created earlier.

```

$clientcertpassword = "1234"

$vpnClientConfiguration = New-AzVpnClientConfiguration ` 
    -ResourceGroupName $resourceGroupName ` 
    -Name $vpnName ` 
    -AuthenticationMethod EAPTLS

Invoke-WebRequest ` 
    -Uri $vpnClientConfiguration.VpnProfileSASUrl ` 
    -OutFile "$vpnTemp\vpnclientconfiguration.zip"

Expand-Archive ` 
    -Path "$vpnTemp\vpnclientconfiguration.zip" ` 
    -DestinationPath "$vpnTemp\vpnclientconfiguration"

$vpnGeneric = "$vpnTemp\vpnclientconfiguration\Generic"
$vpnProfile = ([xml](Get-Content -Path "$vpnGeneric\VpnSettings.xml")).VpnProfile

$exportedclientcertpath = $vpnTemp + "P2SClientCert.pfx"
$clientcertname = "CN=" + $vpnProfile.VpnServer

$clientcert = New-SelfSignedCertificate ` 
    -Type Custom ` 
    -DnsName $vpnProfile.VpnServer ` 
    -KeySpec Signature ` 
    -Subject $clientcertname ` 
    -KeyExportPolicy Exportable ` 
    -HashAlgorithm sha256 ` 
    -KeyLength 2048 ` 
    -CertStoreLocation $certLocation ` 
    -Signer $rootcert ` 
    -TextExtension @("2.5.29.37={text}1.3.6.1.5.5.7.3.2")

$mypwd = ConvertTo-SecureString -String $clientcertpassword -Force -AsPlainText

Export-PfxCertificate ` 
    -FilePath $exportedclientcertpath ` 
    -Password $mypwd ` 
    -Cert $clientcert | Out-Null

```

Configure the VPN client

The Azure virtual network gateway will create a downloadable package with configuration files required to initialize the VPN connection on your on-premises Windows machine. We will configure the VPN connection using the [Always On VPN](#) feature of Windows 10/Windows Server 2016+. This package also contains executable packages which will configure the legacy Windows VPN client, if so desired. This guide uses Always On VPN rather than the legacy Windows VPN client as the Always On VPN client allows end-users to connect/disconnect from the Azure VPN without having administrator permissions to their machine.

The following script will install the client certificate required for authentication against the virtual network gateway, download, and install the VPN package. Remember to replace <computer1> and <computer2> with the desired computers. You can run this script on as many machines as you desire by adding more PowerShell sessions to the \$sessions array. Your user account must be an administrator on each of these machines. If one of these machines is the local machine you are running the script from, you must run the script from an elevated PowerShell session.

```

$sessions = [System.Management.Automation.Runspaces.PSSession[]]@()
$sessions += New-PSSession -ComputerName "<computer1>" 
$sessions += New-PSSession -ComputerName "<computer2>" 

foreach ($session in $sessions) {
    Invoke-Command -Session $session -ArgumentList $vpnTemp -ScriptBlock {

```

```

Invoke-Command -Session $session -ArgumentList $vpnTemp -ScriptBlock {
    $vpnTemp = $args[0]
    if (-Not (Test-Path $vpnTemp)) {
        New-Item `

            -ItemType Directory `

            -Force `

            -Path "C:\vpn-temp" | Out-Null
    }
}

Copy-Item `

    -Path $exportedclientcertpath, $exportedrootcertpath, "$vpnTemp\vpnclientconfiguration.zip" `

    -Destination $vpnTemp `

    -ToSession $session

Invoke-Command `

    -Session $session `

    -ArgumentList `

        $mypwd, `

        $vpnTemp, `

        $virtualNetworkName `

    -ScriptBlock {
        $mypwd = $args[0]
        $vpnTemp = $args[1]
        $virtualNetworkName = $args[2]

        Import-PfxCertificate `

            -Exportable `

            -Password $mypwd `

            -CertStoreLocation "Cert:\LocalMachine\My" `

            -FilePath "$vpnTemp\P2SClientCert.pfx" | Out-Null

        Import-Certificate `

            -FilePath "$vpnTemp\P2SRootCert.cer" `

            -CertStoreLocation "Cert:\LocalMachine\Root" | Out-Null

        Expand-Archive `

            -Path "$vpnTemp\vpnclientconfiguration.zip" `

            -DestinationPath "$vpnTemp\vpnclientconfiguration"
        $vpnGeneric = "$vpnTemp\vpnclientconfiguration\Generic"

        $vpnProfile = ([xml](Get-Content -Path "$vpnGeneric\VpnSettings.xml")).VpnProfile

        Add-VpnConnection `

            -Name $virtualNetworkName `

            -ServerAddress $vpnProfile.VpnServer `

            -TunnelType Ikev2 `

            -EncryptionLevel Required `

            -AuthenticationMethod MachineCertificate `

            -SplitTunneling `

            -AllUserConnection

        Add-VpnConnectionRoute `

            -Name $virtualNetworkName `

            -DestinationPrefix $vpnProfile.Routes `

            -AllUserConnection

        Add-VpnConnectionRoute `

            -Name $virtualNetworkName `

            -DestinationPrefix $vpnProfile.VpnClientAddressPool `

            -AllUserConnection

        rasdial $virtualNetworkName
    }
}

Remove-Item -Path $vpnTemp -Recurse

```

Mount Azure file share

Now that you have set up your Point-to-Site VPN, you can use it to mount the Azure file share on the computers you setup via PowerShell. The following example will mount the share, list the root directory of the share to prove the share is actually mounted, and the unmount the share. Unfortunately, it is not possible to mount the share persistently over PowerShell remoting. To mount persistently, see [Use an Azure file share with Windows](#).

```
$myShareToMount = "<file-share>"  
  
$storageAccountKeys = Get-AzStorageAccountKey `  
    -ResourceGroupName $resourceGroupName `  
    -Name $storageAccountName  
$storageAccountKey = ConvertTo-SecureString `  
    -String $storageAccountKeys[0].Value `  
    -AsPlainText `  
    -Force  
  
$nic = Get-AzNetworkInterface -ResourceId $privateEndpoint.NetworkInterfaces[0].Id  
$storageAccountPrivateIP = $nic.IpConfigurations[0].PrivateIpAddress  
  
Invoke-Command `  
    -Session $sessions `  
    -ArgumentList `  
        $storageAccountName, `  
        $storageAccountKey, `  
        $storageAccountPrivateIP, `  
        $myShareToMount `  
    -ScriptBlock {  
        $storageAccountName = $args[0]  
        $storageAccountKey = $args[1]  
        $storageAccountPrivateIP = $args[2]  
        $myShareToMount = $args[3]  
  
        $credential = [System.Management.Automation.PSCredential]::new(  
            "AZURE\$storageAccountName",  
            $storageAccountKey)  
  
        New-PSDrive `  
            -Name Z `  
            -PSProvider FileSystem `  
            -Root "\$storageAccountPrivateIP\$myShareToMount" `  
            -Credential $credential `  
            -Persist | Out-Null  
        Get-ChildItem -Path Z:\  
        Remove-PSDrive -Name Z  
    }  
}
```

See also

- [Networking considerations for direct Azure file share access](#)
- [Configure a Point-to-Site \(P2S\) VPN on Linux for use with Azure Files](#)
- [Configure a Site-to-Site \(S2S\) VPN for use with Azure Files](#)

Configure a Point-to-Site (P2S) VPN on Linux for use with Azure Files

10/30/2019 • 8 minutes to read • [Edit Online](#)

You can use a Point-to-Site (P2S) VPN connection to mount your Azure file shares over SMB from outside of Azure, without opening up port 445. A Point-to-Site VPN connection is a VPN connection between Azure and an individual client. To use a P2S VPN connection with Azure Files, a P2S VPN connection will need to be configured for each client that wants to connect. If you have many clients that need to connect to your Azure file shares from your on-premises network, you can use a Site-to-Site (S2S) VPN connection instead of a Point-to-Site connection for each client. To learn more, see [Configure a Site-to-Site VPN for use with Azure Files](#).

We strongly recommend that you read [Azure Files networking overview](#) before continuing with this how to article for a complete discussion of the networking options available for Azure Files.

The article details the steps to configure a Point-to-Site VPN on Linux to mount Azure file shares directly on-premises. If you're looking to route Azure File Sync traffic over a VPN, please see [configuring Azure File Sync proxy and firewall settings](#).

Prerequisites

- The most recent version of the Azure CLI. For more information on how to install the Azure CLI, see [Install the Azure PowerShell CLI](#) and select your operating system. If you prefer to use the Azure PowerShell module on Linux, you may, however the instructions below are presented for Azure CLI.
- An Azure file share you would like to mount on-premises. You may use either a [standard](#) or a [premium Azure file share](#) with your Point-to-Site VPN.

Install required software

The Azure virtual network gateway can provide VPN connections using several VPN protocols, including IPsec and OpenVPN. This guide shows how to use IPsec and uses the strongSwan package to provide the support on Linux.

Verified with Ubuntu 18.10.

```
sudo apt install strongswan strongswan-pki libstrongswan-extra-plugins curl libxml2-utils cifs-utils  
installDir="/etc/"
```

Deploy a virtual network

To access your Azure file share and other Azure resources from on-premises via a Point-to-Site VPN, you must create a virtual network, or VNet. The P2S VPN connection you will automatically create is a bridge between your on-premises Linux machine and this Azure virtual network.

The following script will create an Azure virtual network with three subnets: one for your storage account's service endpoint, one for your storage account's private endpoint, which is required to access the storage account on-premises without creating custom routing for the public IP of the storage account that may change, and one for your virtual network gateway that provides the VPN service.

Remember to replace `<region>`, `<resource-group>`, and `<desired-vnet-name>` with the appropriate values for your environment.

```
region=<region>
resourceGroupName=<resource-group>
virtualNetworkName=<desired-vnet-name>

virtualNetwork=$(az network vnet create \
    --resource-group $resourceGroupName \
    --name $virtualNetworkName \
    --location $region \
    --address-prefixes "192.168.0.0/16" \
    --query "newVNet.id" | tr -d ''')

serviceEndpointSubnet=$(az network vnet subnet create \
    --resource-group $resourceGroupName \
    --vnet-name $virtualNetworkName \
    --name "ServiceEndpointSubnet" \
    --address-prefixes "192.168.0.0/24" \
    --service-endpoints "Microsoft.Storage" \
    --query "id" | tr -d '')

privateEndpointSubnet=$(az network vnet subnet create \
    --resource-group $resourceGroupName \
    --vnet-name $virtualNetworkName \
    --name "PrivateEndpointSubnet" \
    --address-prefixes "192.168.1.0/24" \
    --query "id" | tr -d '')

gatewaySubnet=$(az network vnet subnet create \
    --resource-group $resourceGroupName \
    --vnet-name $virtualNetworkName \
    --name "GatewaySubnet" \
    --address-prefixes "192.168.2.0/24" \
    --query "id" | tr -d '')
```

Restrict the storage account to the virtual network

By default when you create a storage account, you can access it from anywhere in the world as long as you have the means to authenticate your request (such as with your Active Directory identity or with the storage account key). To restrict access to this storage account to the virtual network you just created, you need to create a network rule set that allows access within the virtual network and denies all other access.

Restricting the storage account to the virtual network requires the use of a service endpoint. The service endpoint is a networking construct by which the public DNS/public IP can be accessed only from within the virtual network. Since the public IP address is not guaranteed to remain the same, we ultimately want to use a private endpoint rather than a service endpoint for the storage account, however it is not possible to restrict the storage account unless a service endpoint is also exposed.

Remember to replace `<storage-account-name>` with the storage account you want to access.

```
storageAccountName=<storage-account-name>

az storage account network-rule add \
--resource-group $resourceGroupName \
--account-name $storageAccountName \
--subnet $serviceEndpointSubnet > /dev/null

az storage account update \
--resource-group $resourceGroupName \
--name $storageAccountName \
--bypass "AzureServices" \
--default-action "Deny" > /dev/null
```

Create a private endpoint (preview)

Creating a private endpoint for your storage account gives your storage account an IP address within the IP address space of your virtual network. When you mount your Azure file share from on-premises using this private IP address, the routing rules autodefined by the VPN installation will route your mount request to the storage account via the VPN.

```

zoneName="privatelink.file.core.windows.net"

storageAccount=$(az storage account show \
--resource-group $resourceGroupName \
--name $storageAccountName \
--query "id" | tr -d ''')

az resource update \
--ids $privateEndpointSubnet \
--set properties.privateEndpointNetworkPolicies=Disabled > /dev/null

az network private-endpoint create \
--resource-group $resourceGroupName \
--name "$storageAccountName-PrivateEndpoint" \
--location $region \
--subnet $privateEndpointSubnet \
--private-connection-resource-id $storageAccount \
--group-ids "file" \
--connection-name "privateEndpointConnection" > /dev/null

az network private-dns zone create \
--resource-group $resourceGroupName \
--name $zoneName > /dev/null

az network private-dns link vnet create \
--resource-group $resourceGroupName \
--zone-name $zoneName \
--name "$virtualNetworkName-link" \
--virtual-network $virtualNetworkName \
--registration-enabled false > /dev/null

networkInterfaceId=$(az network private-endpoint show \
--name "$storageAccountName-PrivateEndpoint" \
--resource-group $resourceGroupName \
--query 'networkInterfaces[0].id' | tr -d ''')

storageAccountPrivateIP=$(az resource show \
--ids $networkInterfaceId \
--api-version 2019-04-01 \
--query "properties.ipConfigurations[0].properties.privateIPAddress" | tr -d '')

fqdnQuery="properties.ipConfigurations[0].properties.privateLinkConnectionProperties.fqdns[0]"
fqdn=$(az resource show \
--ids $networkInterfaceId \
--api-version 2019-04-01 \
--query $fqdnQuery | tr -d '')

az network private-dns record-set a create \
--name $storageAccountName \
--zone-name $zoneName \
--resource-group $resourceGroupName > /dev/null

```

Create certificates for VPN authentication

In order for VPN connections from your on-premises Linux machines to be authenticated to access your virtual network, you must create two certificates: a root certificate, which will be provided to the virtual machine gateway, and a client certificate, which will be signed with the root certificate. The following script creates the required certificates.

```
rootCertName="P2SRootCert"
username="client"
password="1234"

mkdir temp
cd temp

sudo ipsec pki --gen --outform pem > rootKey.pem
sudo ipsec pki --self --in rootKey.pem --dn "CN=$rootCertName" --ca --outform pem > rootCert.pem

rootCertificate=$(openssl x509 -in rootCert.pem -outform der | base64 -w0 ; echo)

sudo ipsec pki --gen --size 4096 --outform pem > "clientKey.pem"
sudo ipsec pki --pub --in "clientKey.pem" | \
    sudo ipsec pki \
        --issue \
        --cacert rootCert.pem \
        --cakey rootKey.pem \
        --dn "CN=$username" \
        --san $username \
        --flag clientAuth \
        --outform pem > "clientCert.pem"

openssl pkcs12 -in "clientCert.pem" -inkey "clientKey.pem" -certfile rootCert.pem -export -out "client.p12" -password "pass:$password"
```

Deploy virtual network gateway

The Azure virtual network gateway is the service that your on-premises Linux machines will connect to. Deploying this service requires two basic components: a public IP that will identify the gateway to your clients wherever they are in the world and a root certificate you created earlier that will be used to authenticate your clients.

Remember to replace `<desired-vpn-name-here>` with the name you would like for these resources.

NOTE

Deploying the Azure virtual network gateway can take up to 45 minutes. While this resource is being deployed, this bash script will block for the deployment to be completed. This is expected.

```
vpnName=<desired-vpn-name-here>
publicIpAddressName="$vpnName-PublicIP"

publicIpAddress=$(az network public-ip create \
--resource-group $resourceGroupName \
--name $publicIpAddressName \
--location $region \
--sku "Basic" \
--allocation-method "Dynamic" \
--query "publicIp.id" | tr -d ''')

az network vnet-gateway create \
--resource-group $resourceGroupName \
--name $vpnName \
--vnet $virtualNetworkName \
--public-ip-addresses $publicIpAddress \
--location $region \
--sku "VpnGw1" \
--gateway-type "Vpn" \
--vpn-type "RouteBased" \
--address-prefixes "172.16.201.0/24" \
--client-protocol "IkeV2" > /dev/null

az network vnet-gateway root-cert create \
--resource-group $resourceGroupName \
--gateway-name $vpnName \
--name $rootCertName \
--public-cert-data $rootCertificate \
--output none
```

Configure the VPN client

The Azure virtual network gateway will create a downloadable package with configuration files required to initialize the VPN connection on your on-premises Linux machine. The following script will place the certificates you created in the correct spot and configure the `ipsec.conf` file with the correct values from the configuration file in the downloadable package.

```

vpnClient=$(az network vnet-gateway vpn-client generate \
--resource-group $resourceGroupName \
--name $vpnName \
--authentication-method EAPTLS | tr -d '')

curl $vpnClient --output vpnClient.zip
unzip vpnClient.zip

vpnServer=$(xmllint --xpath "string(/VpnProfile/VpnServer)" Generic/VpnSettings.xml)
vpnType=$(xmllint --xpath "string(/VpnProfile/VpnType)" Generic/VpnSettings.xml | tr '[:upper:]' '[:lower:]')
routes=$(xmllint --xpath "string(/VpnProfile/Routes)" Generic/VpnSettings.xml)

sudo cp "${installDir}ipsec.conf" "${installDir}ipsec.conf.backup"
sudo cp "Generic/VpnServerRoot.cer" "${installDir}ipsec.d/cacerts"
sudo cp "${username}.p12" "${installDir}ipsec.d/private"

echo -e "\nconn $virtualNetworkName" | sudo tee -a "${installDir}ipsec.conf" > /dev/null
echo -e "\tkeyexchange=$vpnType" | sudo tee -a "${installDir}ipsec.conf" > /dev/null
echo -e "\ttype=tunnel" | sudo tee -a "${installDir}ipsec.conf" > /dev/null
echo -e "\tleftfirewall=yes" | sudo tee -a "${installDir}ipsec.conf" > /dev/null
echo -e "\tleft=%any" | sudo tee -a "${installDir}ipsec.conf" > /dev/null
echo -e "\tleftauth=eap-tls" | sudo tee -a "${installDir}ipsec.conf" > /dev/null
echo -e "\tleftid=%client" | sudo tee -a "${installDir}ipsec.conf" > /dev/null
echo -e "\tright=$vpnServer" | sudo tee -a "${installDir}ipsec.conf" > /dev/null
echo -e "\trightid=%$vpnServer" | sudo tee -a "${installDir}ipsec.conf" > /dev/null
echo -e "\trightsubnet=$routes" | sudo tee -a "${installDir}ipsec.conf" > /dev/null
echo -e "\tleftsourceip=%config" | sudo tee -a "${installDir}ipsec.conf" > /dev/null
echo -e "\tauto=add" | sudo tee -a "${installDir}ipsec.conf" > /dev/null

echo ": P12 client.p12 '$password'" | sudo tee -a "${installDir}ipsec.secrets" > /dev/null

sudo ipsec restart
sudo ipsec up $virtualNetworkName

```

Mount Azure file share

Now that you have set up your Point-to-Site VPN, you can mount your Azure file share. The following example will mount the share non-persistently. To mount persistently, see [Use an Azure file share with Linux](#).

```

fileShareName="myshare"

mntPath="/mnt/$storageAccountName/$fileShareName"
sudo mkdir -p $mntPath

storageAccountKey=$(az storage account keys list \
--resource-group $resourceGroupName \
--account-name $storageAccountName \
--query "[0].value" | tr -d '')

smbPath="//$storageAccountPrivateIP/$fileShareName"
sudo mount -t cifs $smbPath $mntPath -o
vers=3.0,username=$storageAccountName,password=$storageAccountKey,serverino

```

See also

- [Azure Files networking overview](#)
- [Configure a Point-to-Site \(P2S\) VPN on Windows for use with Azure Files](#)
- [Configure a Site-to-Site \(S2S\) VPN for use with Azure Files](#)

Manage registered servers with Azure File Sync

2/25/2020 • 7 minutes to read • [Edit Online](#)

Azure File Sync allows you to centralize your organization's file shares in Azure Files without giving up the flexibility, performance, and compatibility of an on-premises file server. It does this by transforming your Windows Servers into a quick cache of your Azure file share. You can use any protocol available on Windows Server to access your data locally (including SMB, NFS, and FTPS) and you can have as many caches as you need across the world.

The following article illustrates how to register and manage a server with a Storage Sync Service. See [How to deploy Azure File Sync](#) for information on how to deploy Azure File Sync end-to-end.

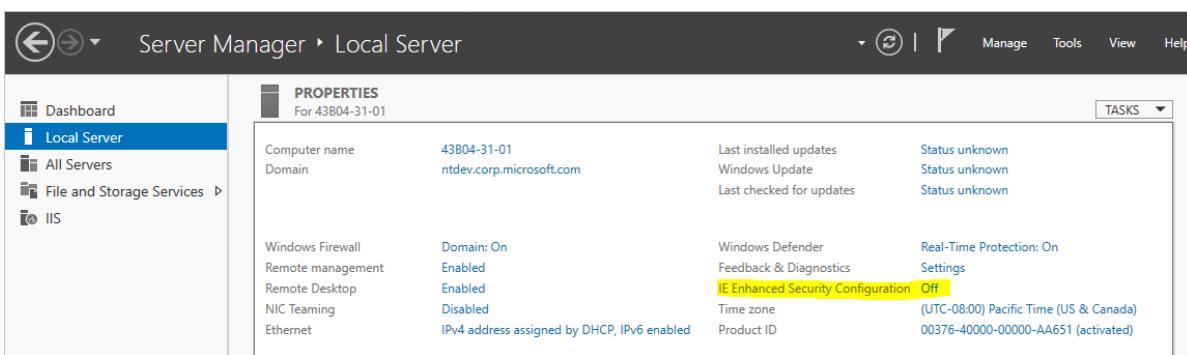
Register/unregister a server with Storage Sync Service

Registering a server with Azure File Sync establishes a trust relationship between Windows Server and Azure. This relationship can then be used to create *server endpoints* on the server, which represent specific folders that should be synced with an Azure file share (also known as a *cloud endpoint*).

Prerequisites

To register a server with a Storage Sync Service, you must first prepare your server with the necessary prerequisites:

- Your server must be running a supported version of Windows Server. For more information, see [Azure File Sync system requirements and interoperability](#).
- Ensure that a Storage Sync Service has been deployed. For more information on how to deploy a Storage Sync Service, see [How to deploy Azure File Sync](#).
- Ensure that the server is connected to the internet and that Azure is accessible.
- Disable the IE Enhanced Security Configuration for administrators with the Server Manager UI.



- Ensure that the Azure PowerShell module is installed on your server. If your server is a member of a Failover Cluster, every node in the cluster will require the Az module. More details on how to install the Az module can be found on the [Install and configure Azure PowerShell](#).

NOTE

We recommend using the newest version of the Az PowerShell module to register/unregister a server. If the Az package has been previously installed on this server (and the PowerShell version on this server is 5.* or greater), you can use the `Update-Module` cmdlet to update this package.

- If you utilize a network proxy server in your environment, configure proxy settings on your server for the sync agent to utilize.
 1. Determine your proxy IP address and port number
 2. Edit these two files:
 - C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Config\machine.config
 - C:\Windows\Microsoft.NET\Framework\v4.0.30319\Config\machine.config
 3. Add the lines in figure 1 (beneath this section) under /System.ServiceModel in the above two files changing 127.0.0.1:8888 to the correct IP address (replace 127.0.0.1) and correct port number (replace 8888):
 4. Set the WinHTTP proxy settings via command line:
 - Show the proxy: netsh winhttp show proxy
 - Set the proxy: netsh winhttp set proxy 127.0.0.1:8888
 - Reset the proxy: netsh winhttp reset proxy
 - if this is setup after the agent is installed, then restart our sync agent: net stop filesyncsvc

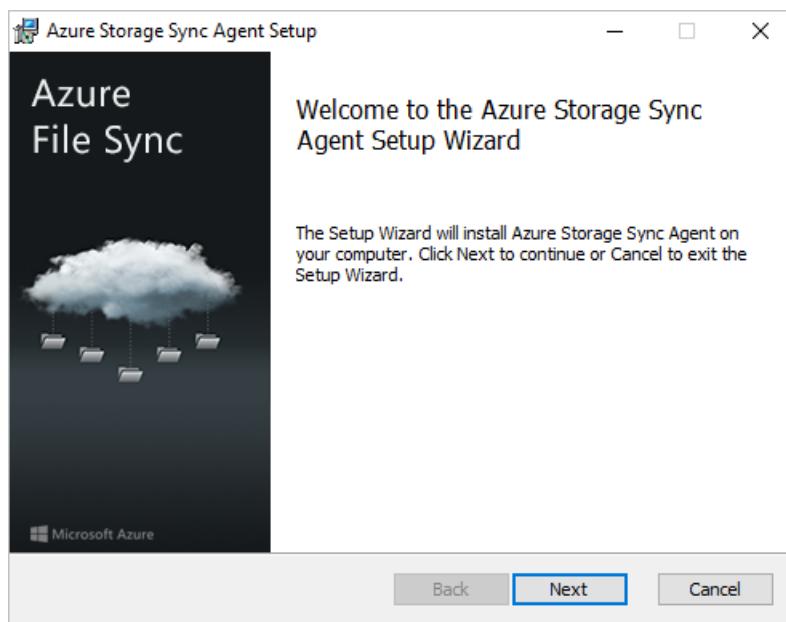
```
Figure 1:
<system.net>
  <defaultProxy enabled="true" useDefaultCredentials="true">
    <proxy autoDetect="false" bypassOnLocal="false" proxyAddress="http://127.0.0.1:8888"
usesystemDefault="false" />
  </defaultProxy>
</system.net>
```

Register a server with Storage Sync Service

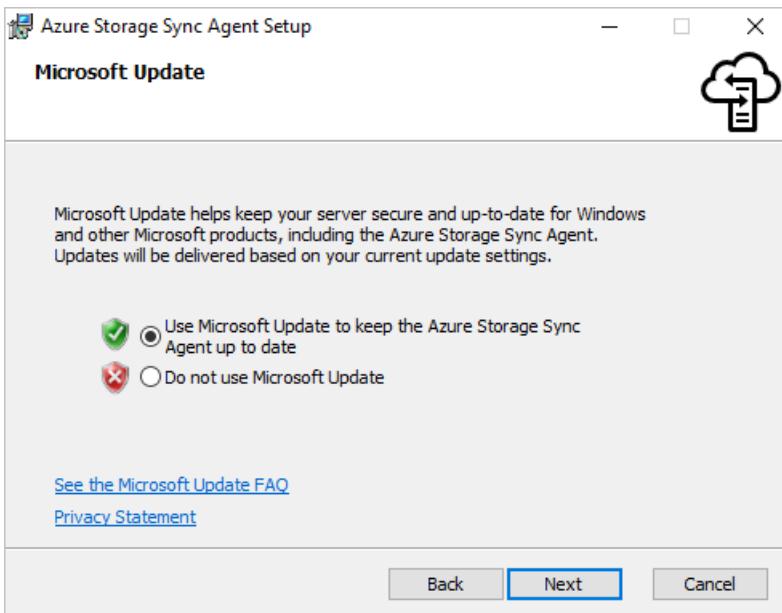
Before a server can be used as a *server endpoint* in an Azure File Sync *sync group*, it must be registered with a *Storage Sync Service*. A server can only be registered with one Storage Sync Service at a time.

Install the Azure File Sync agent

1. [Download the Azure File Sync agent](#).
2. Start the Azure File Sync agent installer.



3. Be sure to enable updates to the Azure File Sync agent using Microsoft Update. It is important because critical security fixes and feature enhancements to the server package are shipped via Microsoft Update.



4. If the server has not been previously registered, the server registration UI will pop up immediately after completing the installation.

IMPORTANT

If the server is a member of a Failover Cluster, the Azure File Sync agent needs to be installed on every node in the cluster.

Register the server using the server registration UI

IMPORTANT

Cloud Solution Provider (CSP) subscriptions cannot use the server registration UI. Instead, use PowerShell (below this section).

1. If the server registration UI did not start immediately after completing the installation of the Azure File Sync agent, it can be started manually by executing
`C:\Program Files\Azure\StorageSyncAgent\ServerRegistration.exe`.
2. Click *Sign-in* to access your Azure subscription.



Sign in and register this server

Sign in to Azure to register with an existing Storage Sync Service.
To create a new Storage Sync Service, go to the Azure portal.

Sign in

3. Pick the correct subscription, resource group, and Storage Sync Service from the dialog.

Choose a Storage Sync Service

Azure Subscription

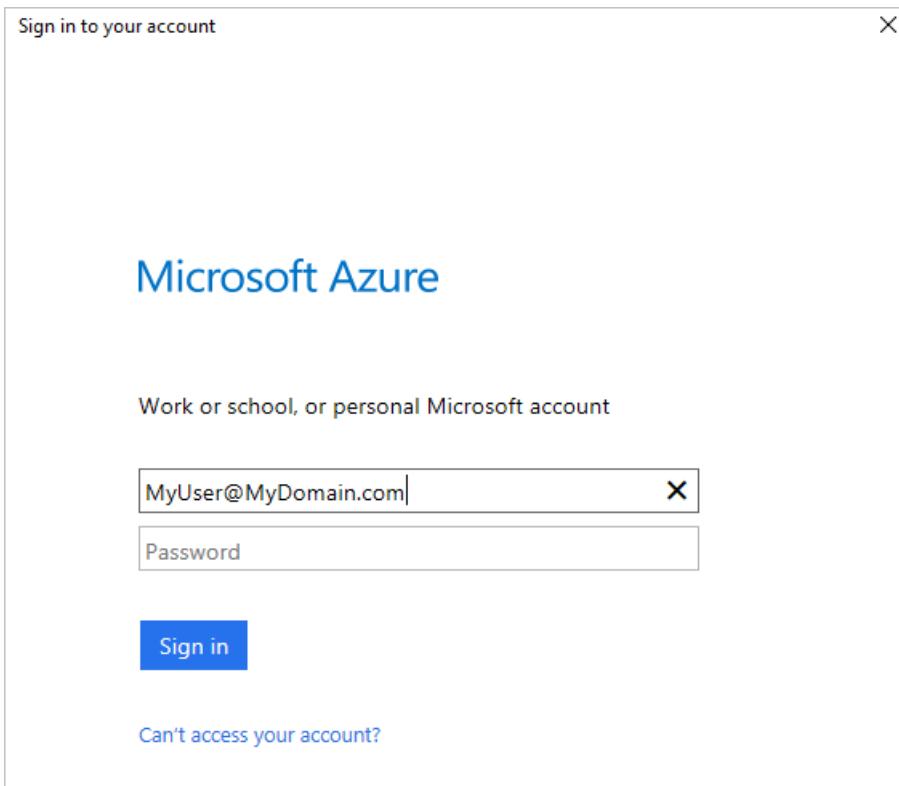
Subscription ID:

Resource Group

Storage Sync Service

Register

4. In preview, one more sign-in is required to complete the process.



IMPORTANT

If the server is a member of a Failover Cluster, each server needs to run the Server Registration. When you view the registered servers in the Azure Portal, Azure File Sync automatically recognizes each node as a member of the same Failover Cluster, and groups them together appropriately.

Register the server with PowerShell

You can also perform server registration via PowerShell. This is the only supported way of server registration for Cloud Solution Provider (CSP) subscriptions:

```
Register-AzStorageSyncServer -ResourceGroupName "<your-resource-group-name>" -StorageSyncServiceName "<your-storage-sync-service-name>"
```

Unregister the server with Storage Sync Service

There are several steps that are required to unregister a server with a Storage Sync Service. Let's take a look at how to properly unregister a server.

WARNING

Do not attempt to troubleshoot issues with sync, cloud tiering, or any other aspect of Azure File Sync by unregistering and registering a server, or removing and recreating the server endpoints unless explicitly instructed to by a Microsoft engineer. Unregistering a server and removing server endpoints is a destructive operation, and tiered files on the volumes with server endpoints will not be "reconnected" to their locations on the Azure file share after the registered server and server endpoints are recreated, which will result in sync errors. Also note, tiered files that exist outside of a server endpoint namespace may be permanently lost. Tiered files may exist within server endpoints even if cloud tiering was never enabled.

(Optional) Recall all tiered data

If you would like files that are currently tiered to be available after removing Azure File Sync (i.e. this is a production, not a test, environment), recall all files on each volume containing server endpoints. Disable cloud tiering for all server endpoints, and then run the following PowerShell cmdlet:

```
Import-Module "C:\Program Files\Azure\StorageSyncAgent\StorageSync.Management.ServerCmdlets.dll"
Invoke-StorageSyncFileRecall -Path <a-volume-with-server-endpoints-on-it>
```

WARNING

If the local volume hosting the server endpoint does not have enough free space to recall all the tiered data, the `Invoke-StorageSyncFileRecall` cmdlet will fail.

Remove the server from all sync groups

Before unregistering the server on the Storage Sync Service, all server endpoints on that server must be removed. This can be done via the Azure portal:

1. Navigate to the Storage Sync Service where your server is registered.
2. Remove all server endpoints for this server in each sync group in the Storage Sync Service. This can be accomplished by right-clicking the relevant server endpoint in the sync group pane.

The screenshot shows the Azure Storage Sync Service interface. At the top, there's a header for a 'Sync group' named 'sidpptest5'. Below the header are buttons for 'Add cloud endpoint', 'Add server endpoint', 'Refresh', and 'Delete'. A large number '1' indicates 'cloud endpoints', with 'sidpptest5' listed below it and a green checkmark in the 'PROVISIONING STATE' column. Below this, a section titled '3 server endpoints' lists two entries: 'ANDREY-CV2.redmond.co.' and 'defenderServer.redmond...'. A context menu is open over the second entry, showing options like 'Properties' and 'Delete'. The 'Delete' option is highlighted with a dashed blue border.

This can also be accomplished with a simple PowerShell script:

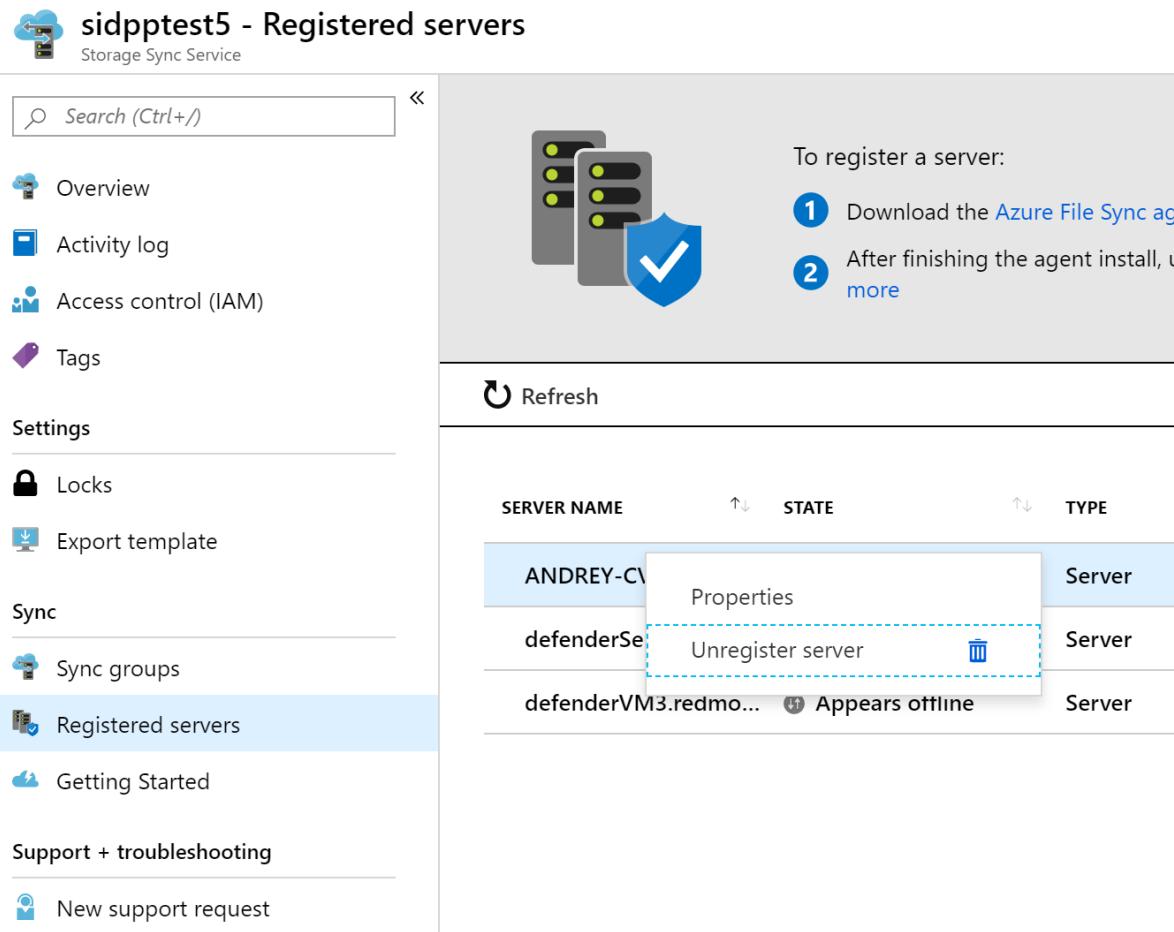
```
Connect-AzAccount
```

```
$storageSyncServiceName = "<your-storage-sync-service>"  
$resourceGroup = "<your-resource-group>"  
  
Get-AzStorageSyncGroup -ResourceGroupName $resourceGroup -StorageSyncServiceName $storageSyncServiceName |  
ForEach-Object {  
    $syncGroup = $_;  
    Get-AzStorageSyncServerEndpoint -ParentObject $syncGroup | Where-Object { $_.ServerEndpointName -eq  
$env:ComputerName } | ForEach-Object {  
        Remove-AzStorageSyncServerEndpoint -InputObject $_  
    }  
}
```

Unregister the server

Now that all data has been recalled and the server has been removed from all sync groups, the server can be unregistered.

1. In the Azure portal, navigate to the *Registered servers* section of the Storage Sync Service.
2. Right-click on the server you want to unregister and click "Unregister Server".



The screenshot shows the 'Registered servers' blade in the Azure Storage Sync Service. The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Settings (Locks, Export template), Sync (Sync groups, Registered servers), and Getting Started. The 'Registered servers' link is highlighted. The main area displays a table of registered servers:

SERVER NAME	STATE	TYPE
ANDREY-CV	Properties	Server
defenderSe	Unregister server	Server
defenderVM3.redmo...	Appears offline	Server

A context menu is open over the second server row, showing options: Properties, Unregister server, and a delete icon.

Ensuring Azure File Sync is a good neighbor in your datacenter

Since Azure File Sync will rarely be the only service running in your datacenter, you may want to limit the network and storage usage of Azure File Sync.

IMPORTANT

Setting limits too low will impact the performance of Azure File Sync synchronization and recall.

Set Azure File Sync network limits

You can throttle the network utilization of Azure File Sync by using the `StorageSyncNetworkLimit` cmdlets.

NOTE

Network limits do not apply when a tiered file is accessed or the `Invoke-StorageSyncFileRecall` cmdlet is used.

For example, you can create a new throttle limit to ensure that Azure File Sync does not use more than 10 Mbps between 9 am and 5 pm (17:00h) during the work week:

```
Import-Module "C:\Program Files\Azure\StorageSyncAgent\StorageSync.Management.ServerCmdlets.dll"
New-StorageSyncNetworkLimit -Day Monday, Tuesday, Wednesday, Thursday, Friday -StartHour 9 -EndHour 17 -
    LimitKbps 10000
```

You can see your limit by using the following cmdlet:

```
Get-StorageSyncNetworkLimit # assumes StorageSync.Management.ServerCmdlets.dll is imported
```

To remove network limits, use `Remove-StorageSyncNetworkLimit`. For example, the following command removes all network limits:

```
Get-StorageSyncNetworkLimit | ForEach-Object { Remove-StorageSyncNetworkLimit -Id $_.Id } # assumes
StorageSync.Management.ServerCmdlets.dll is imported
```

Use Windows Server storage QoS

When Azure File Sync is hosted in a virtual machine running on a Windows Server virtualization host, you can use Storage QoS (storage quality of service) to regulate storage IO consumption. The Storage QoS policy can be set either as a maximum (or limit, like how `StorageSyncNetwork` limit is enforced above) or as a minimum (or reservation). Setting a minimum instead of a maximum allows Azure File Sync to burst to use available storage bandwidth if other workloads are not using it. For more information, see [Storage Quality of Service](#).

See also

- [Planning for an Azure File Sync deployment](#)
- [Deploy Azure File Sync](#)
- [Monitor Azure File Sync](#)
- [Troubleshoot Azure File Sync](#)

Add/remove an Azure File Sync server endpoint

9/12/2019 • 4 minutes to read • [Edit Online](#)

Azure File Sync allows you to centralize your organization's file shares in Azure Files without giving up the flexibility, performance, and compatibility of an on-premises file server. It does this by transforming your Windows Servers into a quick cache of your Azure file share. You can use any protocol available on Windows Server to access your data locally (including SMB, NFS, and FTPS) and you can have as many caches as you need across the world.

A *server endpoint* represents a specific location on a *registered server*, such as a folder on a server volume or the root of the volume. Multiple server endpoints can exist on the same volume if their namespaces are not overlapping (for example, F:\sync1 and F:\sync2). You can configure cloud tiering policies individually for each server endpoint. If you add a server location with an existing set of files as a server endpoint to a sync group, those files will be merged with any other files already on other endpoints in the sync group.

See [How to deploy Azure File Sync](#) for information on how to deploy Azure File Sync end-to-end.

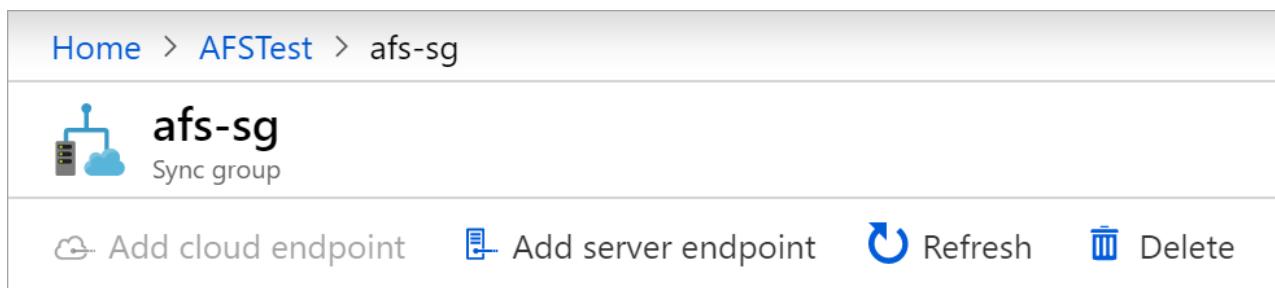
Prerequisites

To create a server endpoint, you must first ensure that the following criteria are met:

- The server has the Azure File Sync agent installed and has been registered. Instructions for installing the Azure File Sync Agent can be found in the [Register/unregister a server with Azure File Sync](#) article.
- Ensure that a Storage Sync Service has been deployed. See [How to deploy Azure File Sync](#) for details on how to deploy a Storage Sync Service.
- Ensure that a sync group has been deployed. Learn how to [Create a sync group](storage-sync-files-deployment-guide.md#create-a-sync-group-and-a-cloud-endpoint).
- Ensure that the server is connected to the internet and that Azure is accessible. We use port 443 for all communication between the server and our service.

Add a server endpoint

To add a server endpoint, navigate to the desired sync group, and select "Add server endpoint".



The screenshot shows a navigation bar with 'Home > AFSTest > afs-sg'. Below this is a card for 'afs-sg' labeled 'Sync group'. At the bottom of the card are four buttons: 'Add cloud endpoint' (with a cloud icon), 'Add server endpoint' (with a server icon), 'Refresh' (with a circular arrow icon), and 'Delete' (with a trash bin icon).

The following information is required under **Add server endpoint**:

- **Registered server:** The name of the server or cluster to create the server endpoint on.
- **Path:** The path on the Windows Server to be synchronized as part of the sync group.
- **Cloud Tiering:** A switch to enable or disable cloud tiering. When enabled, cloud tiering will *tier* files to your Azure file shares. This converts on-premises file shares into a cache, rather than a complete copy of the dataset, to help you manage space efficiency on your server.
- **Volume Free Space:** the amount of free space to reserve on the volume which the server endpoint resides. For

example, if the volume free space is set to 50% on a volume with a single server endpoint, roughly half the amount of data will be tiered to Azure Files. Regardless of whether cloud tiering is enabled, your Azure file share always has a complete copy of the data in the sync group.

Select **Create** to add the server endpoint. The files within a namespace of a sync group will now be kept in sync.

Remove a server endpoint

If you desire to discontinue using Azure File Sync for a given server endpoint, you can remove the server endpoint.

WARNING

Do not attempt to troubleshoot issues with sync, cloud tiering, or any other aspect of Azure File Sync by removing and recreating the server endpoint unless explicitly instructed to by a Microsoft engineer. Removing a server endpoint is a destructive operation, and tiered files within the server endpoint will not be "reconnected" to their locations on the Azure file share after the server endpoint is recreated, which will result in sync errors. Also note, tiered files that exist outside of the server endpoint namespace may be permanently lost. Tiered files may exist within your server endpoint even if cloud tiering was never enabled.

To ensure that all tiered files are recalled before removing the server endpoint, disable cloud tiering on the server endpoint, and then execute the following PowerShell cmdlet to recall all tiered files within your server endpoint namespace:

```
Import-Module "C:\Program Files\Azure\StorageSyncAgent\StorageSync.Management.ServerCmdlets.dll"
Invoke-StorageSyncFileRecall -Path <path-to-to-your-server-endpoint> -Order CloudTieringPolicy
```

Specifying `-Order CloudTieringPolicy` will recall the most recently modified files first. Other optional but useful parameters to consider are:

- `-ThreadCount` determines the how many files can be recalled in parallel.
- `-PerFileRetryCount` determines how often a recall will be attempted of a file that is currently blocked.
- `-PerFileRetryDelaySeconds` determines the time in seconds between retry to recall attempts and should always be used in combination with the previous parameter.

NOTE

If the local volume hosting the server does not have enough free space to recall all the tiered data, the `Invoke-StorageSyncFileRecall` cmdlet fails.

To remove the server endpoint:

1. Navigate to the Storage Sync Service where your server is registered.
2. Navigate to the desired sync group.
3. Remove the server endpoint you desire in the sync group in the Storage Sync Service. This can be accomplished by right-clicking the relevant server endpoint in the sync group pane.

The screenshot shows the Azure File Sync management interface. At the top, there's a header with a sync group icon and the name "sidpptest5". Below the header are buttons for "Add cloud endpoint", "Add server endpoint", "Refresh", and "Delete".

Cloud Endpoints: A section titled "1 cloud endpoints" lists one entry: "sidpptest5" under "AZURE FILE SHARE". The "PROVISIONING STATE" column shows a green checkmark.

Server Endpoints: A section titled "3 server endpoints" lists two entries: "ANDREY-CV2.redmond.co." and "defenderServer.redmond...". The "HEALTH" column shows a blue bar for ANDREY-CV2.redmond.co. The "FILES NOT SYNCING" column shows a blue bar for defenderServer.redmond... A context menu is open over the "defenderServer.redmond..." entry, with options "Properties" and "Delete".

Next steps

- Register/unregister a server with Azure File Sync
- Planning for an Azure File Sync deployment
- Monitor Azure File Sync

Managing Storage in the Azure independent clouds using PowerShell

12/5/2019 • 4 minutes to read • [Edit Online](#)

Most people use Azure Public Cloud for their global Azure deployment. There are also some independent deployments of Microsoft Azure for reasons of sovereignty and so on. These independent deployments are referred to as "environments." The following list details the independent clouds currently available.

- [Azure Government Cloud](#)
- [Azure China 21Vianet Cloud operated by 21Vianet in China](#)
- [Azure German Cloud](#)

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](#).

Using an independent cloud

To use Azure Storage in one of the independent clouds, you connect to that cloud instead of Azure Public. To use one of the independent clouds rather than Azure Public:

- You specify the *environment* to which to connect.
- You determine and use the available regions.
- You use the correct endpoint suffix, which is different from Azure Public.

The examples require Azure PowerShell module Az version 0.7 or later. In a PowerShell window, run

```
Get-Module -ListAvailable Az
```

to find the version. If nothing is listed, or you need to upgrade, see [Install Azure PowerShell module](#).

Log in to Azure

Run the [Get-AzEnvironment](#) cmdlet to see the available Azure environments:

```
Get-AzEnvironment
```

Sign in to your account that has access to the cloud to which you want to connect and set the environment. This example shows how to sign into an account that uses the Azure Government Cloud.

```
Connect-AzAccount -Environment AzureUSGovernment
```

To access the China Cloud, use the environment **AzureChinaCloud**. To access the German Cloud, use **AzureGermanCloud**.

At this point, if you need the list of locations to create a storage account or another resource, you can query the locations available for the selected cloud using [Get-AzLocation](#).

```
Get-AzLocation | select Location, DisplayName
```

The following table shows the locations returned for the German cloud.

LOCATION	DISPLAY NAME
germanycentral	Germany Central
germanynortheast	Germany Northeast

Endpoint suffix

The endpoint suffix for each of these environments is different from the Azure Public endpoint. For example, the blob endpoint suffix for Azure Public is **blob.core.windows.net**. For the Government Cloud, the blob endpoint suffix is **blob.core.usgovcloudapi.net**.

Get endpoint using Get-AzEnvironment

Retrieve the endpoint suffix using [Get-AzEnvironment](#). The endpoint is the *StorageEndpointSuffix* property of the environment.

The following code snippets show how to retrieve the endpoint suffix. All of these commands return something like "core.cloudapp.net" or "core.cloudapi.de", etc. Append the suffix to the storage service to access that service. For example, "queue.core.cloudapi.de" will access the queue service in German Cloud.

This code snippet retrieves all of the environments and the endpoint suffix for each one.

```
Get-AzEnvironment | select Name, StorageEndpointSuffix
```

This command returns the following results.

NAME	STORAGEENDPOINTSUFFIX
AzureChinaCloud	core.chinacloudapi.cn
AzureCloud	core.windows.net
AzureGermanCloud	core.cloudapi.de
AzureUSGovernment	core.usgovcloudapi.net

To retrieve all of the properties for the specified environment, call **Get-AzEnvironment** and specify the cloud name. This code snippet returns a list of properties; look for **StorageEndpointSuffix** in the list. The following example is for the German Cloud.

```
Get-AzEnvironment -Name AzureGermanCloud
```

The results are similar to the following values:

```
|Property Name|Value|----|----|| Name | AzureGermanCloud || EnableAdfsAuthentication | False ||  
ActiveDirectoryServiceEndpointResource | http://management.core.cloudapi.de/ || GalleryURL |  
https://gallery.cloudapi.de/ || ManagementPortalUrl | https://portal.microsoftazure.de/ ||  
ServiceManagementUrl | https://manage.core.cloudapi.de/ || PublishSettingsFileUrl|
```

```
https://manage.microsoftazure.de/publishsettings/index || ResourceManagerUrl |
```

```
http://management.microsoftazure.de/ || SqlDatabaseDnsSuffix | .database.cloudapi.de ||
```

StorageEndpointSuffix | core.cloudapi.de || ... | ... | To retrieve just the storage endpoint suffix property, retrieve the specific cloud and ask for just that one property.

```
$environment = Get-AzEnvironment -Name AzureGermanCloud  
Write-Host "Storage EndPoint Suffix = " $environment.StorageEndpointSuffix
```

This command returns the following information:

```
Storage Endpoint Suffix = core.cloudapi.de
```

Get endpoint from a storage account

You can also examine the properties of a storage account to retrieve the endpoints:

```
# Get a reference to the storage account.  
$resourceGroup = "myexistingresourcegroup"  
$storageAccountName = "myexistingstorageaccount"  
$storageAccount = Get-AzStorageAccount `  
    -ResourceGroupName $resourceGroup `  
    -Name $storageAccountName  
    # Output the endpoints.  
Write-Host "blob endpoint = " $storageAccount.PrimaryEndpoints.Blob  
Write-Host "file endpoint = " $storageAccount.PrimaryEndpoints.File  
Write-Host "queue endpoint = " $storageAccount.PrimaryEndpoints.Queue  
Write-Host "table endpoint = " $storageAccount.PrimaryEndpoints.Table
```

For a storage account in the Government Cloud, this command returns the following output:

```
blob endpoint = http://myexistingstorageaccount.blob.core.usgovcloudapi.net/  
file endpoint = http://myexistingstorageaccount.file.core.usgovcloudapi.net/  
queue endpoint = http://myexistingstorageaccount.queue.core.usgovcloudapi.net/  
table endpoint = http://myexistingstorageaccount.table.core.usgovcloudapi.net/
```

After setting the environment

From here going forward, you can use the same PowerShell used to manage your storage accounts and access the data plane as described in the article [Using Azure PowerShell with Azure Storage](#).

Clean up resources

If you created a new resource group and a storage account for this exercise, you can remove both assets by deleting the resource group. Deleting the resource group deletes all resources contained within the group.

```
Remove-AzResourceGroup -Name $resourceGroup
```

Next steps

- [Persisting user logins across PowerShell sessions](#)
- [Azure Government storage](#)
- [Microsoft Azure Government Developer Guide](#)
- [Developer Notes for Azure China 21Vianet Applications](#)
- [Azure Germany Documentation](#)

Change how a storage account is replicated

2/14/2020 • 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 ¹	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 ¹
...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 ¹

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

¹ Incurs a one-time egress charge.

Change the replication setting

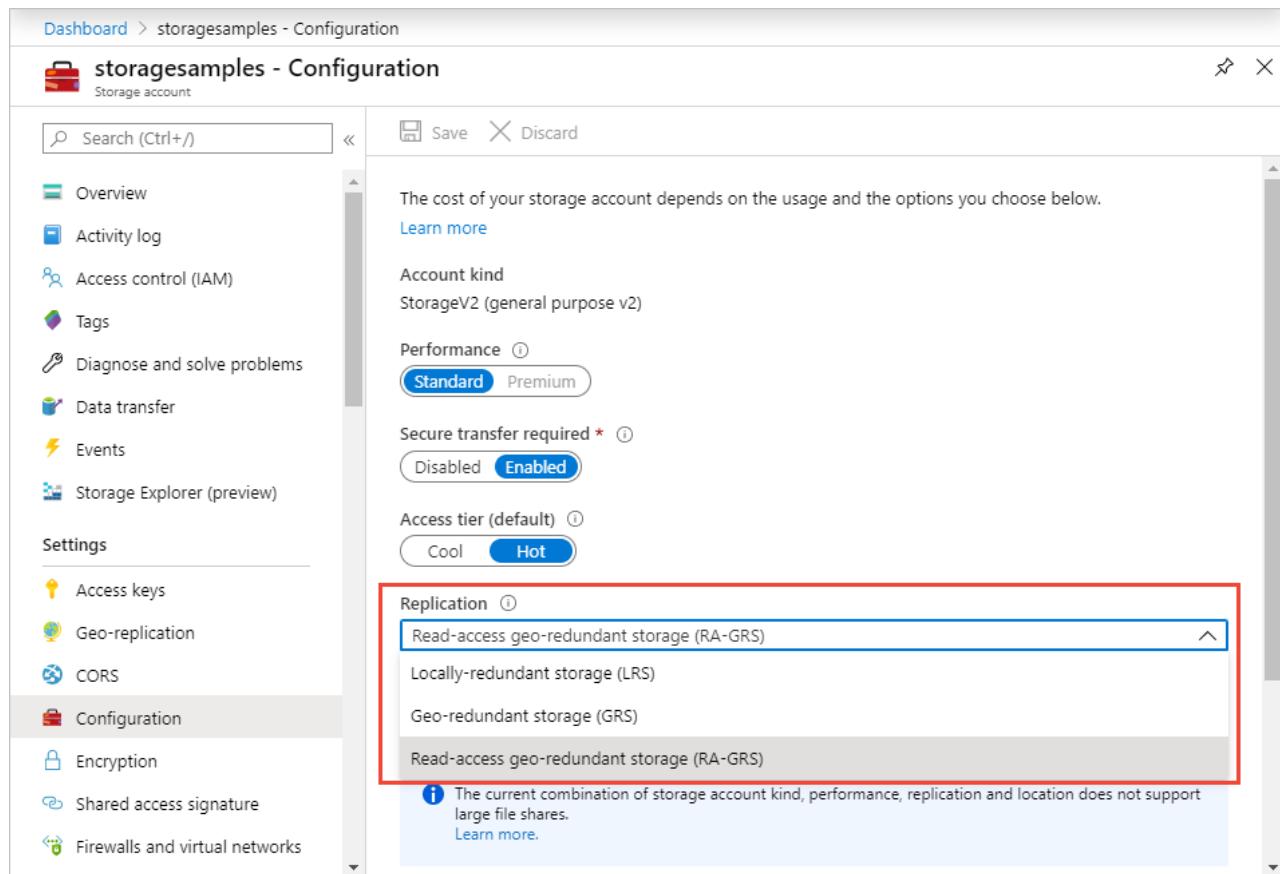
You can use the Azure portal, PowerShell, or Azure CLI to change the replication setting for a storage account, as long as you are not changing how data is replicated in the primary region. If you are migrating from LRS in the primary region to ZRS in the primary region or vice versa, then you must perform 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 account settings like Overview, Activity log, and CORS. The main pane shows the configuration details. In the "Replication" section, the dropdown menu is set to "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 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 cost of your storage account depends on the usage and the options you choose below.

[Learn more](#)

Account kind

Storage (general purpose v1)

 This account can be upgraded to a General Purpose v2 account with additional features. Upgrading is permanent and will result in billing changes. [Learn more](#) 

[Upgrade](#)

Performance 

Standard Premium

* Secure transfer required 

Disabled Enabled

Replication 

Zone-redundant storage (ZRS classic) 

 The replication setting for a storage accounts using ZRS can't be changed.

Data Lake Storage Gen2

Hierarchical namespace 

Disabled Enabled

Costs associated with changing how data is replicated

The costs associated with changing how data is replicated depend on your conversion path. Ordering from least to the most expensive, Azure Storage redundancy offerings include LRS, ZRS, GRS, RA-GRS, GZRS, and RA-GZRS.

For example, going *from* LRS to any other type of replication will incur additional charges because you are moving to a more sophisticated redundancy level. Migrating *to* GRS or RA-GRS will incur an egress bandwidth charge 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](#)

Initiate a storage account failover (preview)

2/12/2020 • 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 steps:

- Register for the account failover preview. For information about how to register, see [About the preview](#).
- 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:

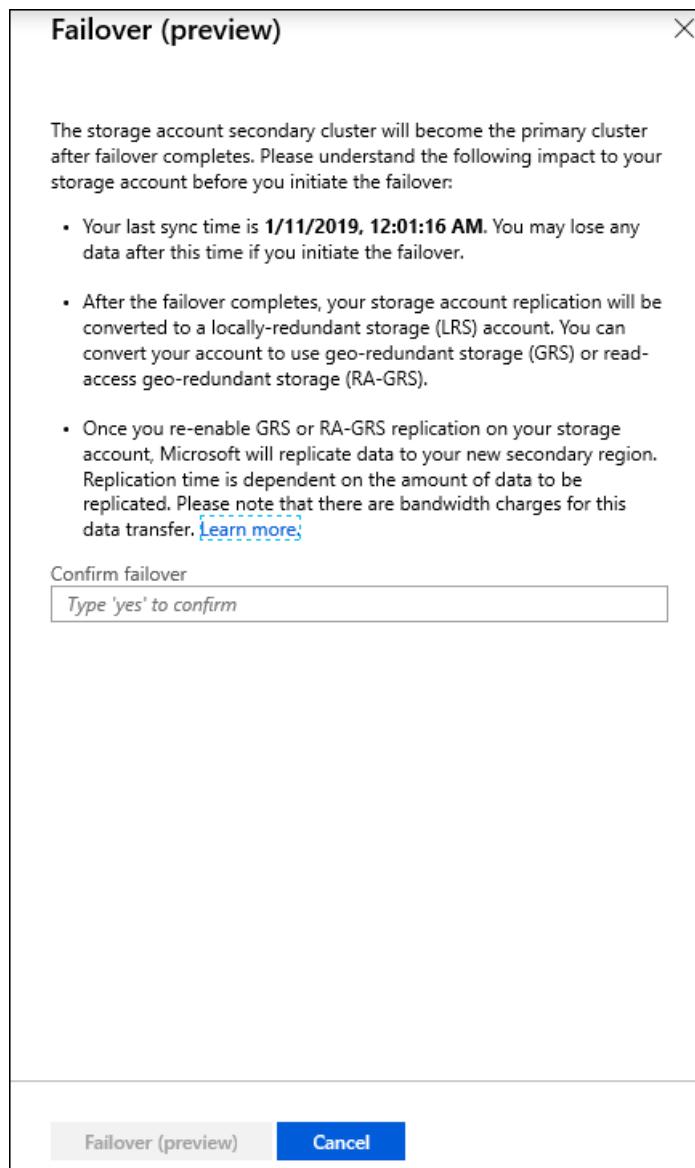
1. Navigate to your storage account.
2. Under **Settings**, select **Geo-replication**. The following image shows the geo-replication and failover status of a storage account.

The screenshot shows the 'Geo-replication' settings page for a storage account. On the left, there's a sidebar with various navigation options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Events, Storage Explorer (preview), and Settings. Under Settings, 'Geo-replication' is selected. The main area displays a world map with two locations marked: 'Primary location' (US West 2) and 'Secondary location' (US West Central). Below the map is a table with the following data:

LOCATION	DATA CENTER TYPE	STATUS	FAILOVER
US West 2	Primary	Available	-
US West Central	Secondary	Available	-

At the bottom of the page is a blue button labeled '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](#)

Monitor Azure File Sync

7/31/2019 • 7 minutes to read • [Edit Online](#)

Use Azure File Sync to centralize your organization's file shares in Azure Files, while keeping the flexibility, performance, and compatibility of an on-premises file server. Azure File Sync transforms Windows Server into a quick cache of your Azure file share. You can use any protocol that's available on Windows Server to access your data locally, including SMB, NFS, and FTPS. You can have as many caches as you need across the world.

This article describes how to monitor your Azure File Sync deployment by using Azure Monitor, Storage Sync Service and Windows Server.

The following monitoring options are currently available.

Azure Monitor

Use [Azure Monitor](#) to view metrics and to configure alerts for sync, cloud tiering, and server connectivity.

Metrics

Metrics for Azure File Sync are enabled by default and are sent to Azure Monitor every 15 minutes.

To view Azure File Sync metrics in Azure Monitor, select the **Storage Sync Services** resource type.

The following metrics for Azure File Sync are available in Azure Monitor:

METRIC NAME	DESCRIPTION
Bytes synced	Size of data transferred (upload and download). Unit: Bytes Aggregation Type: Sum Applicable dimensions: Server Endpoint Name, Sync Direction, Sync Group Name
Cloud tiering recall	Size of data recalled. Note: This metric will be removed in the future. Use the Cloud tiering recall size metric to monitor size of data recalled. Unit: Bytes Aggregation Type: Sum Applicable dimension: Server Name
Cloud tiering recall size	Size of data recalled. Unit: Bytes Aggregation Type: Sum Applicable dimension: Server Name, Sync Group Name
Cloud tiering recall size by application	Size of data recalled by application. Unit: Bytes Aggregation Type: Sum Applicable dimension: Application Name, Server Name, Sync Group Name

METRIC NAME	DESCRIPTION
Cloud tiering recall throughput	Size of data recall throughput. Unit: Bytes Aggregation Type: Sum Applicable dimension: Server Name, Sync Group Name
Files not syncing	Count of files that are failing to sync. Unit: Count Aggregation Type: Sum Applicable dimensions: Server Endpoint Name, Sync Direction, Sync Group Name
Files synced	Count of files transferred (upload and download). Unit: Count Aggregation Type: Sum Applicable dimensions: Server Endpoint Name, Sync Direction, Sync Group Name
Server online status	Count of heartbeats received from the server. Unit: Count Aggregation Type: Maximum Applicable dimension: Server Name
Sync session result	Sync session result (1=successful sync session; 0=failed sync session) Unit: Count Aggregation Types: Maximum Applicable dimensions: Server Endpoint Name, Sync Direction, Sync Group Name

Alerts

To configure alerts in Azure Monitor, select the Storage Sync Service and then select the [Azure File Sync metric](#) to use for the alert.

The following table lists some example scenarios to monitor and the proper metric to use for the alert:

SCENARIO	METRIC TO USE FOR ALERT
Server endpoint health in the portal = Error	Sync session result
Files are failing to sync to a server or cloud endpoint	Files not syncing
Registered server is failing to communicate with the Storage Sync Service	Server online status
Cloud tiering recall size has exceeded 500GiB in a day	Cloud tiering recall size

To learn more about configuring alerts in Azure Monitor, see [Overview of alerts in Microsoft Azure](#).

Storage Sync Service

To view registered server health, server endpoint health, and metrics, go to the Storage Sync Service in the Azure

portal. You can view registered server health in the **Registered servers** blade and server endpoint health in the **Sync groups** blade.

Registered server health

- If the **Registered server** state is **Online**, the server is successfully communicating with the service.
- If the **Registered server** state is **Appears Offline**, verify that the Storage Sync Monitor (AzureStorageSyncMonitor.exe) process on the server is running. If the server is behind a firewall or proxy, see [this article](#) to configure the firewall and proxy.

Server endpoint health

- The server endpoint health in the portal is based on the sync events that are logged in the Telemetry event log on the server (ID 9102 and 9302). If a sync session fails because of a transient error, such as error canceled, sync might still appear healthy in the portal as long as the current sync session is making progress. Event ID 9302 is used to determine if files are being applied. For more information, see [sync health](#) and [sync progress](#).
- If the portal shows a sync error because sync is not making progress, see the [troubleshooting documentation](#) for guidance.

Metric charts

- The following metric charts are viewable in the Storage Sync Service portal:

METRIC NAME	DESCRIPTION	BLADE NAME
Bytes synced	Size of data transferred (upload and download)	Sync group, Server endpoint
Cloud tiering recall	Size of data recalled	Registered servers
Files not syncing	Count of files that are failing to sync	Server endpoint
Files synced	Count of files transferred (upload and download)	Sync group, Server endpoint
Server online status	Count of heartbeats received from the server	Registered servers

- To learn more, see [Azure Monitor](#).

NOTE

The charts in the Storage Sync Service portal have a time range of 24 hours. To view different time ranges or dimensions, use Azure Monitor.

Windows Server

On Windows Server, you can view cloud tiering, registered server, and sync health.

Event logs

Use the Telemetry event log on the server to monitor registered server, sync, and cloud tiering health. The Telemetry event log is located in Event Viewer under *Applications and Services\Microsoft\FileSync\Agent*.

Sync health:

- Event ID 9102 is logged after a sync session finishes. Use this event to determine if sync sessions are successful (**HResult = 0**) and if there are per-item sync errors. For more information, see the [sync health](#)

and [per-item errors](#) documentation.

NOTE

Sometimes sync sessions fail overall or have a non-zero PerItemErrorCode. However, they still make forward progress, and some files sync successfully. You can see this in the Applied fields such as AppliedItemCount, AppliedDirCount, AppliedTombstoneCount, and AppliedSizeBytes. These fields tell you how much of the session succeeded. If you see multiple sync sessions fail in a row, and they have an increasing Applied count, give sync time to try again before you open a support ticket.

- Event ID 9302 is logged every 5 to 10 minutes if there's an active sync session. Use this event to determine if the current sync session is making progress (**AppliedItemCount > 0**). If sync is not making progress, the sync session should eventually fail, and an Event ID 9102 will be logged with the error. For more information, see the [sync progress documentation](#).

Registered server health:

- Event ID 9301 is logged every 30 seconds when a server queries the service for jobs. If GetNextJob finishes with **status = 0**, the server is able to communicate with the service. If GetNextJob finishes with an error, check the [troubleshooting documentation](#) for guidance.

Cloud tiering health:

- To monitor tiering activity on a server, use Event ID 9003, 9016 and 9029 in the Telemetry event log, which is located in Event Viewer under *Applications and Services\Microsoft\FileSync\Agent*.
 - Event ID 9003 provides error distribution for a server endpoint. For example: Total Error Count and ErrorCode. One event is logged per error code.
 - Event ID 9016 provides ghosting results for a volume. For example: Free space percent is, Number of files ghosted in session, and Number of files failed to ghost.
 - Event ID 9029 provides ghosting session information for a server endpoint. For example: Number of files attempted in the session, Number of files tiered in the session, and Number of files already tiered.
- To monitor recall activity on a server, use Event ID 9005, 9006, 9009 and 9059 in the Telemetry event log, which is located in Event Viewer under *Applications and Services\Microsoft\FileSync\Agent*.
 - Event ID 9005 provides recall reliability for a server endpoint. For example: Total unique files accessed, and Total unique files with failed access.
 - Event ID 9006 provides recall error distribution for a server endpoint. For example: Total Failed Requests, and ErrorCode. One event is logged per error code.
 - Event ID 9009 provides recall session information for a server endpoint. For example: DurationSeconds, CountFilesRecallSucceeded, and CountFilesRecallFailed.
 - Event ID 9059 provides application recall distribution for a server endpoint. For example: ShareId, Application Name, and TotalEgressNetworkBytes.

Performance counters

Use the Azure File Sync performance counters on the server to monitor sync activity.

To view Azure File Sync performance counters on the server, open Performance Monitor (Perfmon.exe). You can find the counters under the **AFS Bytes Transferred** and **AFS Sync Operations** objects.

The following performance counters for Azure File Sync are available in Performance Monitor:

PERFORMANCE OBJECT\ COUNTER NAME	DESCRIPTION
AFS Bytes Transferred\Downloaded Bytes/sec	Number of bytes downloaded per second.

PERFORMANCE OBJECT\ COUNTER NAME	DESCRIPTION
AFS Bytes Transferred\Uploaded Bytes/sec	Number of bytes uploaded per second.
AFS Bytes Transferred\Total Bytes/sec	Total bytes per second (upload and download).
AFS Sync Operations\Downloaded Sync Files/sec	Number of files downloaded per second.
AFS Sync Operations\Uploaded Sync Files/sec	Number of files uploaded per second.
AFS Sync Operations\Total Sync File Operations/sec	Total number of files synced (upload and download).

Next steps

- [Planning for an Azure File Sync deployment](#)
- [Consider firewall and proxy settings](#)
- [Deploy Azure File Sync](#)
- [Troubleshoot Azure File Sync](#)
- [Azure Files frequently asked questions](#)

Migrate bulk data to Azure File Sync

1/20/2020 • 7 minutes to read • [Edit Online](#)

You can migrate bulk data to Azure File Sync in two ways:

- **Upload your files by using Azure File Sync.** This is the simplest method. Move your files locally to Windows Server 2012 R2 or later, and install the Azure File Sync agent. After you set up the sync, your files will be uploaded from the server. (Our customers currently experience an average upload speed of 1 TiB about every two days.) To ensure that your server doesn't use too much of the bandwidth for your datacenter, you might want to set up a [bandwidth throttling schedule](#).
- **Transfer your files offline.** If you don't have enough bandwidth, you might not be able to upload files to Azure in a reasonable amount of time. The challenge is the initial sync of the whole set of files. To overcome this challenge, use offline bulk migration tools such as the [Azure Data Box family](#).

This article explains how to migrate files offline in a way that's compatible with Azure File Sync. Follow these instructions to avoid file conflicts and to preserve your file and folder access control lists (ACLs) and timestamps after you enable the sync.

Migration tools

The process we describe in this article works not only for Data Box but also for other offline migration tools. It also works for tools such as AzCopy, Robocopy, or partner tools and services that work straight over the internet. However to overcome the initial upload challenge, follow the steps in this article to use these tools in a way that's compatible with Azure File Sync.

In some cases you need to move from one Windows Server to another Windows Server before adopting Azure File Sync. [Storage Migration Service](#) (SMS) can help with that. Whether you need to migrate to a Server OS version that is supported by Azure File Sync (Windows Server 2012R2 and up) or you simply need to migrate because you are buying a new system for Azure File Sync, SMS has numerous features and advantages that will help get your migration done smoothly.

Benefits of using a tool to transfer data offline

Here are the main benefits of using a transfer tool like Data Box for offline migration:

- You don't have to upload all of your files over the network. For large namespaces, this tool could save significant network bandwidth and time.
- When you use Azure File Sync, no matter which transfer tool you use (Data Box, Azure Import/Export service, and so on), your live server uploads only the files that change after you move the data to Azure.
- Azure File Sync syncs your file and folder ACLs even if the offline bulk migration tool doesn't transport ACLs.
- Data Box and Azure File Sync require no downtime. When you use Data Box to transfer data into Azure, you use network bandwidth efficiently and preserve the file fidelity. You also keep your namespace up to date by uploading only the files that change after you move the data to Azure.

Prerequisites for the offline data transfer

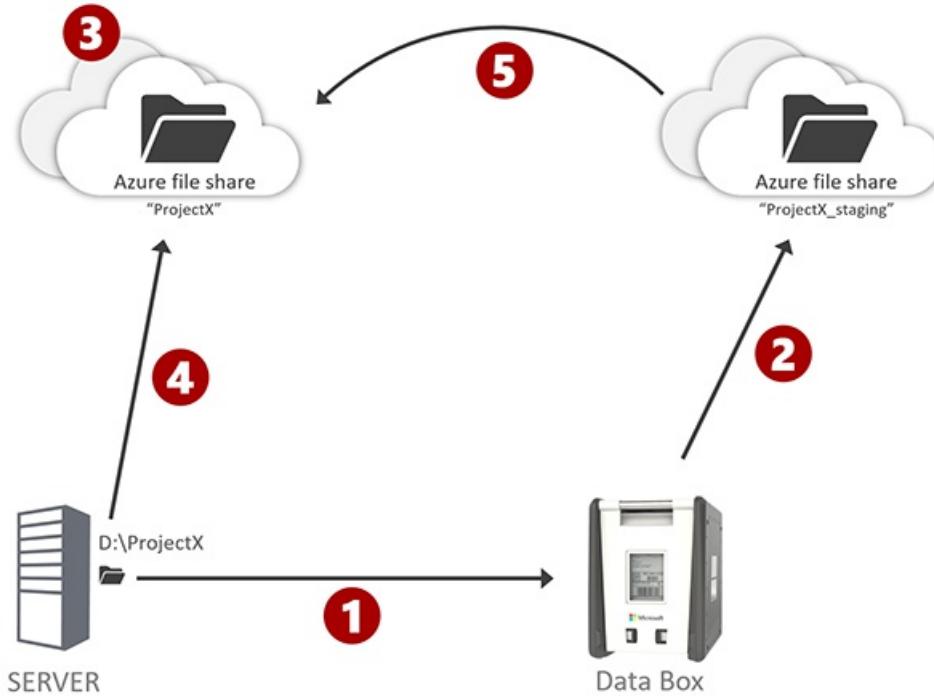
You should not enable sync on the server you are migrating before you complete your offline data transfer. Other things to consider before you begin are as follows:

- If you plan to use Data Box for your bulk migration: Review the [deployment prerequisites for Data Box](#).

- Plan your final Azure File Sync topology: [Plan for an Azure File Sync deployment](#)
- Select Azure storage account(s) that will hold the file shares you want to sync with. Ensure that your bulk migration happens to temporary staging shares in the same Storage Account(s). Bulk migration can only be enabled utilizing a final- and a staging- share that reside in the same storage account.
- A bulk migration can only be utilized when you create a new sync relationship with a server location. You can't enable a bulk migration with an existing sync relationship.

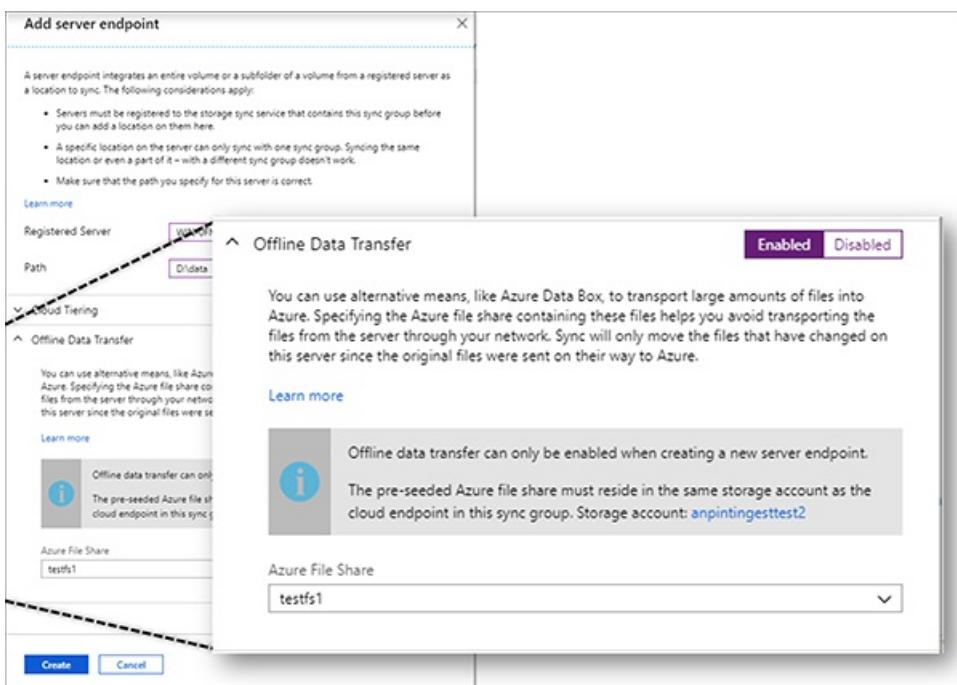
Process for offline data transfer

Here's how to set up Azure File Sync in a way that's compatible with bulk migration tools such as Azure Data Box:



STEP	DETAIL
1	Order your Data Box. The Data Box family offers several products to meet your needs. When you receive your Data Box, follow its documentation to copy your data to this UNC path on the Data Box: <DeviceIPAddres><StorageAccountName_AzFile><ShareName>. Here, ShareName is the name of the staging share. Send the Data Box back to Azure.
2	Wait until your files show up in the Azure file shares that you chose as temporary staging shares. <i>Do not enable syncing to these shares.</i>
3	Create a new empty share for each file share that Data Box created for you. This new share should be in the same storage account as the Data Box share. How to create a new Azure file share .
4	Create a sync group in a storage sync service. Reference the empty share as a cloud endpoint. Repeat this step for every Data Box file share. Set up Azure File Sync .

STEP	DETAIL
5	<p>Add your live server directory as a server endpoint. In the process, specify that you moved the files to Azure, and reference the staging shares. You can enable or disable cloud tiering as needed. While creating a server endpoint on your live server, reference the staging share. On the Add server endpoint blade, under Offline Data Transfer, select Enabled, and then select the staging share that must be in the same storage account as the cloud endpoint. Here, the list of available shares is filtered by storage account and shares that aren't already syncing.</p>



Syncing the share

After you create your server endpoint, the sync will start. The sync process determines whether each file on the server also exists in the staging share where Data Box deposited the files. If the file exists there, the sync process copies the file from the staging share rather than uploading it from the server. If the file doesn't exist in the staging share, or if a newer version is available on the local server, the sync process uploads the file from the local server.

IMPORTANT

You can enable the bulk migration mode only while you're creating a server endpoint. After you establish a server endpoint, you can't integrate bulk-migrated data from an already syncing server into the namespace.

ACLs and timestamps on files and folders

Azure File Sync ensures that file and folder ACLs are synced from the live server even if the bulk migration tool that you used didn't initially transport ACLs. Because of this, the staging share doesn't need to contain any ACLs for files and folders. When you enable the offline data migration feature as you create a new server endpoint, all file ACLs are synced on the server. Newly created and modified timestamps are also synced.

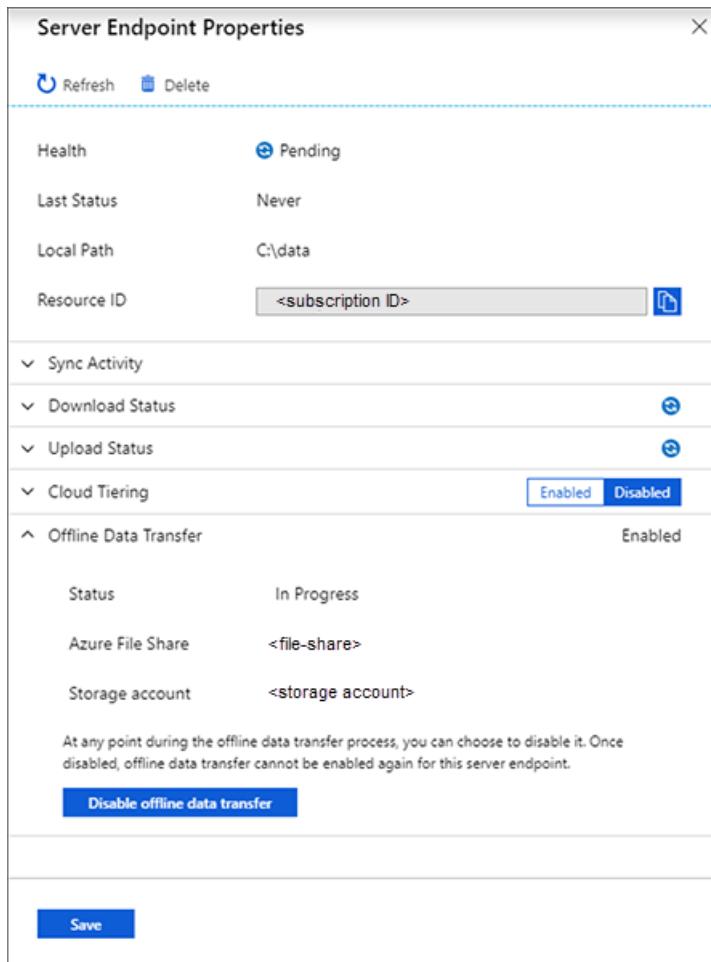
Shape of the namespace

When you enable the sync, the server's contents determine the shape of the namespace. If files are deleted from the local server after the Data Box snapshot and migration finish, these files don't move into the live, syncing

namespace. They stay in the staging share, but they aren't copied. This is necessary because the sync keeps the namespace according to the live server. The Data Box *snapshot* is just a staging ground for efficient file copy. It's not the authority for the shape of the live namespace.

Cleaning up after bulk migration

As the server completes its initial sync of the namespace, the Data Box bulk-migrated files use the staging file share. On the **Server Endpoint Properties** blade in Azure portal, in the **Offline Data Transfer** section, the status changes from **In Progress** to **Completed**.



Now you can clean up the staging share to save costs:

1. On the **Server Endpoint Properties** blade, when the status is **Completed**, select **Disable offline data transfer**.
2. Consider deleting the staging share to save costs. The staging share probably doesn't contain file and folder ACLs, so it's not very useful. For backup point-in-time purposes, create a real [snapshot of the syncing Azure file share](#). You can [set up Azure Backup to take snapshots](#) on a schedule.

Disable the offline data transfer mode only when the state is **Completed** or when you want to cancel because of a misconfiguration. If you disable the mode during a deployment, files will start to upload from the server even if your staging share is still available.

IMPORTANT

After you disable the offline data transfer mode, you can't enable it again, even if the staging share from the bulk migration is still available.

Next steps

- [Plan for an Azure File Sync deployment](#)
- [Deploy Azure File Sync](#)

StorSimple migration to Azure File Sync

12/5/2019 • 4 minutes to read • [Edit Online](#)

StorSimple is a discontinued Microsoft product. Extended support for this product and its cloud service will last until the end of 2022. It is important to start planning for a migration off of StorSimple right away.

The default and strategic long-term Azure service StorSimple appliances can be migrated to, is Azure File Sync. It is a generally available Azure service with a super-set of features over StorSimple.

Full cloud-side migration with limited downtime

This article describes the concept of how a migration will commence. It is imperative to note that customers in need of migration off of StorSimple and onto Azure File Sync do not need to proceed on their own.

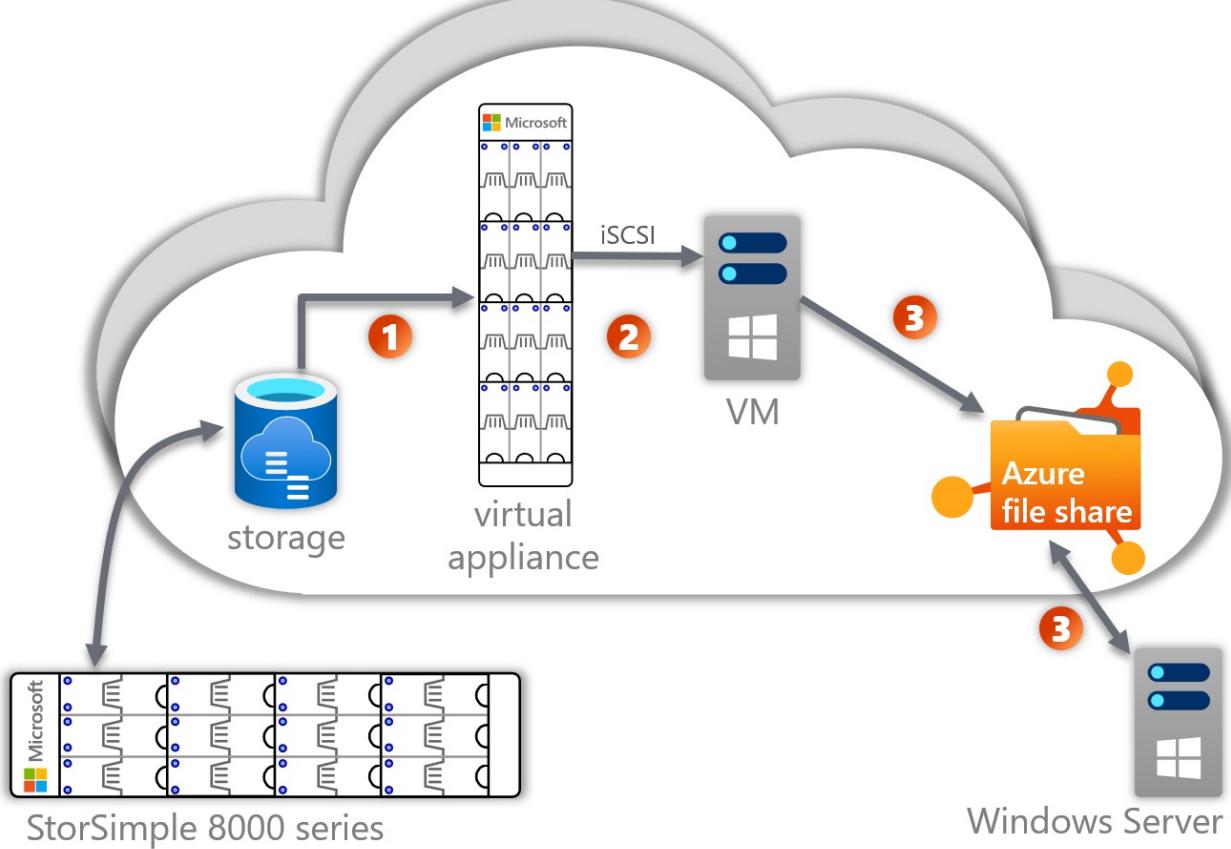
IMPORTANT

Microsoft is committed to assist customers in their migration. Email AzureFiles@microsoft.com for a customized migration plan as well as assistance during the migration.

Migration approach

The migration to Azure File Sync will commence cloud-side with minimal impact to on-premises and limited downtime. The below concept is targeted at StorSimple 8000 series appliances. If you are in need of migration from StorSimple 7000 series, the first step entails a free upgrade to a matching 8000 series loaner device from Microsoft. Reach out to AzureFiles@microsoft.com and we will help you organize an appropriate number of loaner devices.

General approach



1. Take a volume clone of your on-premises StorSimple appliance and mount it to a temporary StorSimple virtual appliance.
2. Connect the virtual appliance via iSCSI to a temporary Azure VM.
3. Install Azure File Sync on the temporary Windows Server VM - a specific registry key for this migration also needs to be set before sync is configured on the server.
 - Depending on the number of shares on your StorSimple volume, deploy just as many Azure file shares. (We recommend deploying one Azure file share per storage account.)
 - Configure sync between the individual share on the Windows Server cloud VM and an Azure file share. (1:1 mapping)
 - Optionally, add a local server as an on-premises performance cache, with cloud tiering enabled. This step is necessary if you wish to replace your on-premises StorSimple with a more feature rich, local cache, powered by Windows Server and Azure File Sync's cloud tiering. Your on-premises Windows Server can be a physical machine or cluster, or a virtual machine. It does not have to have as much storage deployed as the dataset size. It only needs enough storage to locally cache the most frequently accessed files.

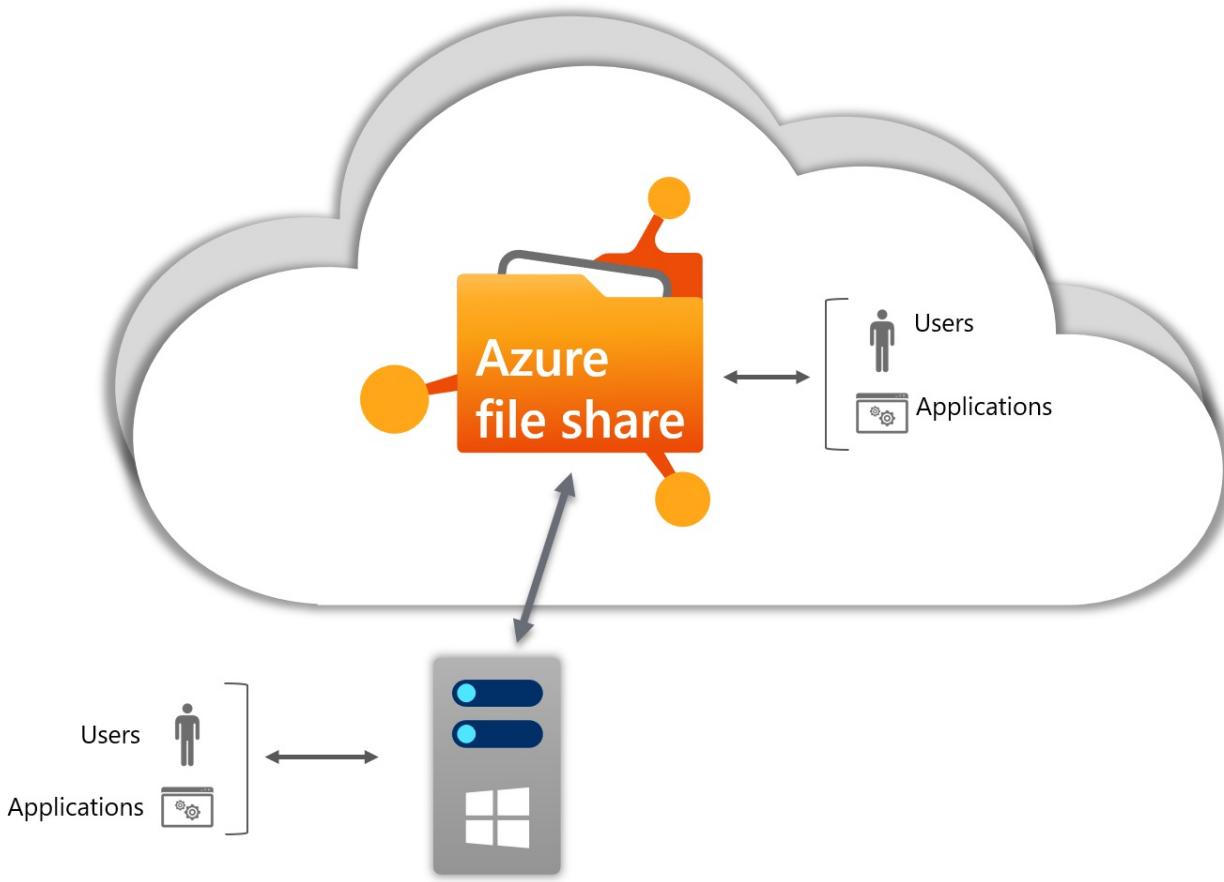
Minimizing downtime

After step 3 above, the StorSimple on-premises appliance is still actively used by users and applications. So the set of files that synced from the initial volume clone is slightly outdated at the time sync completes. The approach to minimizing downtime is to repeat the sync from volume clone process, such that sync finishes faster and faster with each iteration, which in turn is enabled by the changes between volume clones become fewer and fewer. You can repeat this process, until the sync from a volume clone can finish in the amount of time you find acceptable for downtime. Once that is the case, block users and applications from making any changes to your StorSimple appliance. Downtime begins. Take another volume clone and let it sync to the server(s) connected. Establish access to your users and applications to your new, Azure File Sync backed, Windows Server. Consider deploying/adjusting a DFS-Namespace to make cutting over from the old StorSimple appliance to your new Windows Server transparent to apps and users. Your migration is complete.

Migration goal

After the migration is complete, the temporary StorSimple virtual appliance and Azure VM can be deprovisioned.

Furthermore the StorSimple on-premises appliance can be deprovisioned as your users and applications are already accessing the Windows Server instead. What you are left with is depicted in the image below. A standard Azure File Sync deployment features a number of Azure file shares and Windows Servers connected to them via Azure File Sync. Remember that a single server can connect different local folders to different file shares at the same time. Also, one Azure file share can sync to many different servers, in the case you need data cached in branch offices. Also check if you can optimize your cloud tiering policies for more efficient use of your on-premises storage space.



Next steps

Familiarize yourself with Azure Files and Azure File Sync. It is important to understand the Azure File Sync terminology and deployment pattern for a successful migration. There is more detailed information available for every step in this overview article. Make sure to reach out to Microsoft for customized help during planning and execution of your migration.

IMPORTANT

Microsoft is committed to assist customers in their migration. Email AzureFiles@microsoft.com for a customized migration plan as well as assistance during the migration.

Additional resources

Azure File Sync, as the target service, has two fundamental documents that we recommend you read, if you are new to Azure File Sync.

- [Azure File Sync - overview](#)

- [Azure File Sync - deployment guide](#)

Azure Files is a storage service in Azure, offering file shares as a service. No need to pay for or maintain a VM or associated VM storage.

- [Azure Files - overview](#)
- [Azure Files - how to deploy an Azure file share](#)

Configure Azure Storage connection strings

12/23/2019 • 8 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. In the **Settings** section of the storage account overview, 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.

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=Eby8vdM02xNOcqFlqUwJPLl1mEt1CDXJ10UzFT50uSRZ6IFsuFq2UVErCz4I6tq/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 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 \(SAS\)](#)

Develop for Azure Files with .NET

2/12/2020 • 14 minutes to read • [Edit Online](#)

This tutorial demonstrates the basics of using .NET to develop applications that use [Azure Files](#) to store file data. This tutorial creates a simple console application to do basic actions with .NET and Azure Files:

- Get the contents of a file.
- Set the maximum size or *quota* for the file share.
- Create a shared access signature (SAS key) for a file that uses a stored access policy defined on the share.
- Copy a file to another file in the same storage account.
- Copy a file to a blob in the same storage account.
- Use Azure Storage Metrics for troubleshooting.

To learn more about Azure Files, see [What is Azure Files?](#)

TIP

Check out the Azure Storage code samples repository

For easy-to-use end-to-end Azure Storage code samples that you can download and run, please check out our list of [Azure Storage Samples](#).

Understanding the .NET APIs

Azure Files provides two broad approaches to client applications: Server Message Block (SMB) and REST. Within .NET, the `System.IO` and `WindowsAzure.Storage` APIs abstract these approaches.

API	WHEN TO USE	NOTES
System.IO	Your application: <ul style="list-style-type: none">• Needs to read/write files by using SMB• Is running on a device that has access over port 445 to your Azure Files account• Doesn't need to manage any of the administrative settings of the file share	File I/O implemented with Azure Files over SMB is generally the same as I/O with any network file share or local storage device. For an introduction to a number of features in .NET, including file I/O, see the Console Application tutorial.
Microsoft.Azure.Storage.File	Your application: <ul style="list-style-type: none">• Can't access Azure Files by using SMB on port 445 because of firewall or ISP constraints• Requires administrative functionality, such as the ability to set a file share's quota or create a shared access signature	This article demonstrates the use of <code>Microsoft.Azure.Storage.File</code> for file I/O using REST instead of SMB and management of the file share.

Create the console application and obtain the assembly

In Visual Studio, create a new Windows console application. The following steps show you how to create a console application in Visual Studio 2019. The steps are similar in other versions of Visual Studio.

1. Start Visual Studio and select **Create a new project**.
2. In **Create a new project**, choose **Console App (.NET Framework)** for C#, and then select **Next**.
3. In **Configure your new project**, enter a name for the app, and select **Create**.

You can add all the code examples in this tutorial to the `Main()` method of your console application's `Program.cs` file.

You can use the Azure Storage client library in any type of .NET application. These types include an Azure cloud service or web app, and desktop and mobile applications. In this guide, we use a console application for simplicity.

Use NuGet to install the required packages

Refer to these packages in your project to complete this tutorial:

- [Microsoft Azure Storage common library for .NET](#)

This package provides programmatic access to common resources in your storage account.

- [Microsoft Azure Storage Blob library for .NET](#)

This package provides programmatic access to blob resources in your storage account.

- [Microsoft Azure Storage File library for .NET](#)

This package provides programmatic access to file resources in your storage account.

- [Microsoft Azure Configuration Manager library for .NET](#)

This package provides a class for parsing a connection string in a configuration file, wherever your application runs.

You can use NuGet to obtain both packages. Follow these steps:

1. In **Solution Explorer**, right-click your project and choose **Manage NuGet Packages**.
2. In **NuGet Package Manager**, select **Browse**. Then search for and choose **Microsoft.Azure.Storage.Blob**, and then select **Install**.

This step installs the package and its dependencies.

3. Search for and install these packages:

- **Microsoft.Azure.Storage.Common**
- **Microsoft.Azure.Storage.File**
- **Microsoft.Azure.ConfigurationManager**

Save your storage account credentials to the App.config file

Next, save your credentials in your project's `App.config` file. In **Solution Explorer**, double-click `App.config` and edit the file so that it is similar to the following example. Replace `myaccount` with your storage account name and `mykey` with your storage account key.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
    </startup>
    <appSettings>
        <add key="StorageConnectionString"
            value="DefaultEndpointsProtocol=https;AccountName=myaccount;AccountKey=StorageAccountKeyEndingIn==" />
    </appSettings>
</configuration>
```

NOTE

The latest version of the Azure storage emulator does not support Azure Files. Your connection string must target an Azure Storage Account in the cloud to work with Azure Files.

Add using directives

In **Solution Explorer**, open the `Program.cs` file, and add the following using directives to the top of the file.

```
using Microsoft.Azure; // Namespace for Azure Configuration Manager
using Microsoft.Azure.Storage; // Namespace for Storage Client Library
using Microsoft.Azure.Storage.Blob; // Namespace for Azure Blobs
using Microsoft.Azure.Storage.File; // Namespace for Azure Files
```

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](#).

Access the file share programmatically

Next, add the following content to the `Main()` method, after the code shown above, to retrieve the connection string. This code gets a reference to the file we created earlier and outputs its contents.

```
// Create a CloudFileClient object for credentialed access to Azure Files.  
CloudFileClient fileClient = storageAccount.CreateCloudFileClient();  
  
// Get a reference to the file share we created previously.  
CloudFileShare share = fileClient.GetShareReference("logs");  
  
// Ensure that the share exists.  
if (share.Exists())  
{  
    // Get a reference to the root directory for the share.  
    CloudFileDirectory rootDir = share.GetRootDirectoryReference();  
  
    // Get a reference to the directory we created previously.  
    CloudFileDirectory sampleDir = rootDir.GetDirectoryReference("CustomLogs");  
  
    // Ensure that the directory exists.  
    if (sampleDir.Exists())  
    {  
        // Get a reference to the file we created previously.  
        CloudFile file = sampleDir.GetFileReference("Log1.txt");  
  
        // Ensure that the file exists.  
        if (file.Exists())  
        {  
            // Write the contents of the file to the console window.  
            Console.WriteLine(file.DownloadTextAsync().Result);  
        }  
    }  
}  
}
```

Run the console application to see the output.

Set the maximum size for a file share

Beginning with version 5.x of the Azure Storage Client Library, you can set the quota (maximum size) for a file share. You can also check to see how much data is currently stored on the share.

Setting the quota for a share limits the total size of the files stored on the share. If the total size of files on the share exceeds the quota set on the share, clients can't increase the size of existing files. Clients can't create new files, unless those files are empty.

The example below shows how to check the current usage for a share and how to set the quota for the share.

```

// Parse the connection string for the storage account.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    Microsoft.Azure.CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create a CloudFileClient object for credentialated access to Azure Files.
CloudFileClient fileClient = storageAccount.CreateCloudFileClient();

// Get a reference to the file share we created previously.
CloudFileShare share = fileClient.GetShareReference("logs");

// Ensure that the share exists.
if (share.Exists())
{
    // Check current usage stats for the share.
    // Note that the ShareStats object is part of the protocol layer for the File service.
    Microsoft.Azure.Storage.File.Protocol.ShareStats stats = share.GetStats();
    Console.WriteLine("Current share usage: {0} GB", stats.Usage.ToString());

    // Specify the maximum size of the share, in GB.
    // This line sets the quota to be 10 GB greater than the current usage of the share.
    share.Properties.Quota = 10 + stats.Usage;
    share.SetProperties();

    // Now check the quota for the share. Call FetchAttributes() to populate the share's properties.
    share.FetchAttributes();
    Console.WriteLine("Current share quota: {0} GB", share.Properties.Quota);
}

```

Generate a shared access signature for a file or file share

Beginning with version 5.x of the Azure Storage Client Library, you can generate a shared access signature (SAS) for a file share or for an individual file. You can also create a stored access policy on a file share to manage shared access signatures. We recommend creating a stored access policy because it lets you revoke the SAS if it becomes compromised.

The following example creates a stored access policy on a share. The example uses that policy to provide the constraints for a SAS on a file in the share.

```

// Parse the connection string for the storage account.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    Microsoft.Azure.CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create a CloudFileClient object for credentialated access to Azure Files.
CloudFileClient fileClient = storageAccount.CreateCloudFileClient();

// Get a reference to the file share we created previously.
CloudFileShare share = fileClient.GetShareReference("logs");

// Ensure that the share exists.
if (share.Exists())
{
    string policyName = "sampleSharePolicy" + DateTime.UtcNow.Ticks;

    // Create a new stored access policy and define its constraints.
    SharedAccessFilePolicy sharedPolicy = new SharedAccessFilePolicy()
    {
        SharedAccessExpiryTime = DateTime.UtcNow.AddHours(24),
        Permissions = SharedAccessFilePermissions.Read | SharedAccessFilePermissions.Write
    };

    // Get existing permissions for the share.
    FileSharePermissions permissions = share.GetPermissions();

    // Add the stored access policy to the share's policies. Note that each policy must have a unique name.
    permissions.SharedAccessPolicies.Add(policyName, sharedPolicy);
    share.SetPermissions(permissions);

    // Generate a SAS for a file in the share and associate this access policy with it.
    CloudFileDirectory rootDir = share.GetRootDirectoryReference();
    CloudFileDirectory sampleDir = rootDir.GetDirectoryReference("CustomLogs");
    CloudFile file = sampleDir.GetFileReference("Log1.txt");
    string sasToken = file.GetSharedAccessSignature(null, policyName);
    Uri fileSasUri = new Uri(file.StorageUri.PrimaryUri.ToString() + sasToken);

    // Create a new CloudFile object from the SAS, and write some text to the file.
    CloudFile fileSas = new CloudFile(fileSasUri);
    fileSas.UploadText("This write operation is authorized via SAS.");
    Console.WriteLine(fileSas.DownloadText());
}

```

For more information about creating and using shared access signatures, see [How a shared access signature works](#).

Copy files

Beginning with version 5.x of the Azure Storage Client Library, you can copy a file to another file, a file to a blob, or a blob to a file. In the next sections, we demonstrate how to do these copy operations programmatically.

You can also use AzCopy to copy one file to another or to copy a blob to a file or the other way around. See [Get started with AzCopy](#).

NOTE

If you are copying a blob to a file, or a file to a blob, you must use a shared access signature (SAS) to authorize access to the source object, even if you are copying within the same storage account.

Copy a file to another file

The following example copies a file to another file in the same share. Because this copy operation copies between files in the same storage account, you can use Shared Key authentication to do the copy.

```

// Parse the connection string for the storage account.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    Microsoft.Azure.CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create a CloudFileClient object for credentialated access to Azure Files.
CloudFileClient fileClient = storageAccount.CreateCloudFileClient();

// Get a reference to the file share we created previously.
CloudFileShare share = fileClient.GetShareReference("logs");

// Ensure that the share exists.
if (share.Exists())
{
    // Get a reference to the root directory for the share.
    CloudFileDirectory rootDir = share.GetRootDirectoryReference();

    // Get a reference to the directory we created previously.
    CloudFileDirectory sampleDir = rootDir.GetDirectoryReference("CustomLogs");

    // Ensure that the directory exists.
    if (sampleDir.Exists())
    {
        // Get a reference to the file we created previously.
        CloudFile sourceFile = sampleDir.GetFileReference("Log1.txt");

        // Ensure that the source file exists.
        if (sourceFile.Exists())
        {
            // Get a reference to the destination file.
            CloudFile destFile = sampleDir.GetFileReference("Log1Copy.txt");

            // Start the copy operation.
            destFile.StartCopy(sourceFile);

            // Write the contents of the destination file to the console window.
            Console.WriteLine(destFile.DownloadText());
        }
    }
}

```

Copy a file to a blob

The following example creates a file and copies it to a blob within the same storage account. The example creates a SAS for the source file, which the service uses to authorize access to the source file during the copy operation.

```

// Parse the connection string for the storage account.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    Microsoft.Azure.CloudConfigurationManager.GetSetting("StorageConnectionString"));

// Create a CloudFileClient object for credentialated access to Azure Files.
CloudFileClient fileClient = storageAccount.CreateCloudFileClient();

// Create a new file share, if it does not already exist.
CloudFileShare share = fileClient.GetShareReference("sample-share");
share.CreateIfNotExists();

// Create a new file in the root directory.
CloudFile sourceFile = share.GetRootDirectoryReference().GetFileReference("sample-file.txt");
sourceFile.UploadText("A sample file in the root directory.");

// Get a reference to the blob to which the file will be copied.
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();
CloudBlobContainer container = blobClient.GetContainerReference("sample-container");
container.CreateIfNotExists();
CloudBlockBlob destBlob = container.GetBlockBlobReference("sample-blob.txt");

// Create a SAS for the file that's valid for 24 hours.
// Note that when you are copying a file to a blob, or a blob to a file, you must use a SAS
// to authorize access to the source object, even if you are copying within the same
// storage account.
string fileSas = sourceFile.GetSharedAccessSignature(new SharedAccessFilePolicy())
{
    // Only read permissions are required for the source file.
    Permissions = SharedAccessFilePermissions.Read,
    SharedAccessExpiryTime = DateTime.UtcNow.AddHours(24)
};

// Construct the URI to the source file, including the SAS token.
Uri fileSasUri = new Uri(sourceFile.StorageUri.PrimaryUri.ToString() + fileSas);

// Copy the file to the blob.
destBlob.StartCopy(fileSasUri);

// Write the contents of the file to the console window.
Console.WriteLine("Source file contents: {0}", sourceFile.DownloadText());
Console.WriteLine("Destination blob contents: {0}", destBlob.DownloadText());

```

You can copy a blob to a file in the same way. If the source object is a blob, then create a SAS to authorize access to that blob during the copy operation.

Share snapshots

Beginning with version 8.5 of the Azure Storage Client Library, you can create a share snapshot. You can also list or browse share snapshots and delete share snapshots. Share snapshots are read-only so no write operations are allowed on share snapshots.

Create share snapshots

The following example creates a file share snapshot.

```

storageAccount = CloudStorageAccount.Parse(ConnectionString);
fClient = storageAccount.CreateCloudFileClient();
string baseShareName = "myazurefileshare";
CloudFileShare myShare = fClient.GetShareReference(baseShareName);
var snapshotShare = myShare.Snapshot();

```

List share snapshots

The following example lists the share snapshots on a share.

```
var shares = fClient.ListShares(baseShareName, ShareListingDetails.All);
```

Browse files and directories within share snapshots

The following example browses files and directory within share snapshots.

```
CloudFileShare mySnapshot = fClient.GetShareReference(baseShareName, snapshotTime);
var rootDirectory = mySnapshot.GetRootDirectoryReference();
var items = rootDirectory.ListFilesAndDirectories();
```

List shares and share snapshots and restore file shares or files from share snapshots

Taking a snapshot of a file share enables you to recover individual files or the entire the file share in the future.

You can restore a file from a file share snapshot by querying the share snapshots of a file share. You can then retrieve a file that belongs to a particular share snapshot. Use that version to either directly read and compare or to restore.

```
CloudFileShare liveShare = fClient.GetShareReference(baseShareName);
var rootDirOfLiveShare = liveShare.GetRootDirectoryReference();
var dirInLiveShare = rootDirOfLiveShare.GetDirectoryReference(dirName);
var fileInLiveShare = dirInLiveShare.GetFileReference(fileName);

CloudFileShare snapshot = fClient.GetShareReference(baseShareName, snapshotTime);
var rootDirOfSnapshot = snapshot.GetRootDirectoryReference();
var dirInSnapshot = rootDirOfSnapshot.GetDirectoryReference(dirName);
var fileInSnapshot = dirInSnapshot.GetFileReference(fileName);

string sasContainerToken = string.Empty;
SharedAccessFilePolicy sasConstraints = new SharedAccessFilePolicy();
sasConstraints.SharedAccessExpiryTime = DateTime.UtcNow.AddHours(24);
sasConstraints.Permissions = SharedAccessFilePermissions.Read;

//Generate the shared access signature on the container, setting the constraints directly on the signature.
sasContainerToken = fileInSnapshot.GetSharedAccessSignature(sasConstraints);

string sourceUri = (fileInSnapshot.Uri.ToString() + sasContainerToken + "&" +
fileInSnapshot.SnapshotTime.ToString()); ;
fileInLiveShare.StartCopyAsync(new Uri(sourceUri));
```

Delete share snapshots

The following example deletes a file share snapshot.

```
CloudFileShare mySnapshot = fClient.GetShareReference(baseShareName, snapshotTime); mySnapshot.Delete(null,
null, null);
```

Troubleshoot Azure Files by using metrics

Azure Storage Analytics now supports metrics for Azure Files. With metrics data, you can trace requests and diagnose issues.

You can enable metrics for Azure Files from the [Azure portal](#). You can also enable metrics programmatically by calling the Set File Service Properties operation with the REST API or one of its analogs in the Storage Client Library.

The following code example shows how to use the Storage Client Library for .NET to enable metrics for Azure

Files.

First, add the following `using` directives to your `Program.cs` file, along with the ones you added above:

```
using Microsoft.Azure.Storage.File.Protocol;
using Microsoft.Azure.Storage.Shared.Protocol;
```

Although Azure Blobs, Azure Tables, and Azure Queues use the shared `ServiceProperties` type in the `Microsoft.Azure.Storage.Shared.Protocol` namespace, Azure Files uses its own type, the `FileServiceProperties` type in the `Microsoft.Azure.Storage.File.Protocol` namespace. You must reference both namespaces from your code, however, for the following code to compile.

```
// Parse your storage connection string from your application's configuration file.
CloudStorageAccount storageAccount = CloudStorageAccount.Parse(
    Microsoft.Azure.CloudConfigurationManager.GetSetting("StorageConnectionString"));
// Create the File service client.
CloudFileClient fileClient = storageAccount.CreateCloudFileClient();

// Set metrics properties for File service.
// Note that the File service currently uses its own service properties type,
// available in the Microsoft.Azure.Storage.File.Protocol namespace.
fileClient.SetServiceProperties(new FileServiceProperties()
{
    // Set hour metrics
    HourMetrics = new MetricsProperties()
    {
        MetricsLevel = MetricsLevel.ServiceAndApi,
        RetentionDays = 14,
        Version = "1.0"
    },
    // Set minute metrics
    MinuteMetrics = new MetricsProperties()
    {
        MetricsLevel = MetricsLevel.ServiceAndApi,
        RetentionDays = 7,
        Version = "1.0"
    }
});

// Read the metrics properties we just set.
FileServiceProperties serviceProperties = fileClient.GetServiceProperties();
Console.WriteLine("Hour metrics:");
Console.WriteLine(serviceProperties.HourMetrics.MetricsLevel);
Console.WriteLine(serviceProperties.HourMetrics.RetentionDays);
Console.WriteLine(serviceProperties.HourMetrics.Version);
Console.WriteLine();
Console.WriteLine("Minute metrics:");
Console.WriteLine(serviceProperties.MinuteMetrics.MetricsLevel);
Console.WriteLine(serviceProperties.MinuteMetrics.RetentionDays);
Console.WriteLine(serviceProperties.MinuteMetrics.Version);
```

If you encounter any problems, you can refer to [Troubleshoot Azure Files problems in Windows](#).

Next steps

For more information about Azure Files, see the following resources:

Conceptual articles and videos

- [Azure Files: a frictionless cloud SMB file system for Windows and Linux](#)
- [Use Azure Files with Linux](#)

Tooling support for File storage

- [Get started with AzCopy](#)
- [Using the Azure CLI with Azure Storage](#)
- [Troubleshoot Azure Files problems in Windows](#)

Reference

- [Azure Storage APIs for .NET](#)
- [File Service REST API](#)

Blog posts

- [Azure File Storage, now generally available](#)
- [Inside Azure File Storage](#)
- [Introducing Microsoft Azure Files Service](#)
- [Persisting connections to Microsoft Azure Files](#)

Develop for Azure Files with Java

11/8/2019 • 5 minutes to read • [Edit Online](#)

TIP

Check out the Azure Storage code samples repository

For easy-to-use end-to-end Azure Storage code samples that you can download and run, please check out our list of [Azure Storage Samples](#).

About this tutorial

This tutorial will demonstrate the basics of using Java to develop applications or services that use Azure Files to store file data. In this tutorial, we will create a console application and show how to perform basic actions with Java and Azure Files:

- Create and delete Azure file shares
- Create and delete directories
- Enumerate files and directories in an Azure file share
- Upload, download, and delete a file

NOTE

Because Azure Files may be accessed over SMB, it is possible to write applications that access the Azure file share using the standard Java I/O classes. This article will describe how to write applications that use the Azure Storage Java SDK, which uses the [Azure Files REST API](#) to talk to Azure Files.

Create a Java application

To build the samples, you will need the Java Development Kit (JDK) and the [Azure Storage SDK for Java](#). You should also have created an Azure storage account.

Set up your application to use Azure Files

To use the Azure storage APIs, add the following statement to the top of the Java file where you intend to access the storage service from.

```
// Include the following imports to use blob APIs.  
import com.microsoft.azure.storage.*;  
import com.microsoft.azure.storage.file.*;
```

Set up an Azure storage connection string

To use Azure Files, you need to connect to your Azure storage account. The first step would be to configure a connection string, which we'll use to connect to your storage account. Let's define a static variable to do that.

```
// Configure the connection-string with your values
public static final String storageConnectionString =
    "DefaultEndpointsProtocol=http;" +
    "AccountName=your_storage_account_name;" +
    "AccountKey=your_storage_account_key";
```

NOTE

Replace `your_storage_account_name` and `your_storage_account_key` with the actual values for your storage account.

Connecting to an Azure storage account

To connect to your storage account, you need to use the **CloudStorageAccount** object, passing a connection string to its **parse** method.

```
// Use the CloudStorageAccount object to connect to your storage account
try {
    CloudStorageAccount storageAccount = CloudStorageAccount.parse(storageConnectionString);
} catch (InvalidKeyException invalidKey) {
    // Handle the exception
}
```

CloudStorageAccount.parse throws an `InvalidKeyException` so you'll need to put it inside a try/catch block.

Create an Azure file share

All files and directories in Azure Files reside in a container called a **Share**. Your storage account can have as many shares as your account capacity allows. To obtain access to a share and its contents, you need to use an Azure Files client.

```
// Create the Azure Files client.
CloudFileClient fileClient = storageAccount.createCloudFileClient();
```

Using the Azure Files client, you can then obtain a reference to a share.

```
// Get a reference to the file share
CloudFileShare share = fileClient.getShareReference("sampleshare");
```

To actually create the share, use the **createIfNotExists** method of the **CloudFileShare** object.

```
if (share.createIfNotExists()) {
    System.out.println("New share created");
}
```

At this point, **share** holds a reference to a share named **sampleshare**.

Delete an Azure file share

Deleting a share is done by calling the **deleteIfExists** method on a **CloudFileShare** object. Here's sample code that does that.

```

try
{
    // Retrieve storage account from connection-string.
    CloudStorageAccount storageAccount = CloudStorageAccount.parse(storageConnectionString);

    // Create the file client.
    CloudFileClient fileClient = storageAccount.createCloudFileClient();

    // Get a reference to the file share
    CloudFileShare share = fileClient.getShareReference("sampleshare");

    if (share.deleteIfExists()) {
        System.out.println("sampleshare deleted");
    }
} catch (Exception e) {
    e.printStackTrace();
}

```

Create a directory

You can also organize storage by putting files inside subdirectories instead of having all of them in the root directory. Azure Files allows you to create as many directories as your account will allow. The code below will create a subdirectory named **sampledir** under the root directory.

```

//Get a reference to the root directory for the share.
CloudFileDirectory rootDir = share.getRootDirectoryReference();

//Get a reference to the sampledir directory
CloudFileDirectory sampleDir = rootDir.getDirectoryReference("sampledir");

if (sampleDir.createIfNotExists()) {
    System.out.println("sampledir created");
} else {
    System.out.println("sampledir already exists");
}

```

Delete a directory

Deleting a directory is a straightforward task, although it should be noted that you cannot delete a directory that still contains files or other directories.

```

// Get a reference to the root directory for the share.
CloudFileDirectory rootDir = share.getRootDirectoryReference();

// Get a reference to the directory you want to delete
CloudFileDirectory containerDir = rootDir.getDirectoryReference("sampledir");

// Delete the directory
if ( containerDir.deleteIfExists() ) {
    System.out.println("Directory deleted");
}

```

Enumerate files and directories in an Azure file share

Obtaining a list of files and directories within a share is easily done by calling **listFilesAndDirectories** on a CloudFileDirectory reference. The method returns a list of ListFileItem objects which you can iterate on. As an example, the following code will list files and directories inside the root directory.

```
//Get a reference to the root directory for the share.  
CloudFileDirectory rootDir = share.getRootDirectoryReference();  
  
for ( ListFileItem fileItem : rootDir.listFilesAndDirectories() ) {  
    System.out.println(fileItem.getUri());  
}
```

Upload a file

In this section, you'll learn how to upload a file from local storage onto the root directory of a share.

The first step in uploading a file is to obtain a reference to the directory where it should reside. You do this by calling the **getRootDirectoryReference** method of the share object.

```
//Get a reference to the root directory for the share.  
CloudFileDirectory rootDir = share.getRootDirectoryReference();
```

Now that you have a reference to the root directory of the share, you can upload a file onto it using the following code.

```
// Define the path to a local file.  
final String filePath = "C:\\temp\\Readme.txt";  
  
CloudFile cloudFile = rootDir.getFileReference("Readme.txt");  
cloudFile.uploadFromFile(filePath);
```

Download a file

One of the more frequent operations you will perform against Azure Files is to download files. In the following example, the code downloads SampleFile.txt and displays its contents.

```
//Get a reference to the root directory for the share.  
CloudFileDirectory rootDir = share.getRootDirectoryReference();  
  
//Get a reference to the directory that contains the file  
CloudFileDirectory sampleDir = rootDir.getDirectoryReference("sampledir");  
  
//Get a reference to the file you want to download  
CloudFile file = sampleDir.getFileReference("SampleFile.txt");  
  
//Write the contents of the file to the console.  
System.out.println(file.downloadText());
```

Delete a file

Another common Azure Files operation is file deletion. The following code deletes a file named SampleFile.txt stored inside a directory named **sampledir**.

```
// Get a reference to the root directory for the share.  
CloudFileDirectory rootDir = share.getRootDirectoryReference();  
  
// Get a reference to the directory where the file to be deleted is in  
CloudFileDirectory containerDir = rootDir.getDirectoryReference("sampledir");  
  
String filename = "SampleFile.txt"  
CloudFile file;  
  
file = containerDir.getFileReference(filename)  
if ( file.deleteIfExists() ) {  
    System.out.println(filename + " was deleted");  
}
```

Next steps

If you would like to learn more about other Azure storage APIs, follow these links.

- [Azure for Java developers/](#)
- [Azure Storage SDK for Java](#)
- [Azure Storage SDK for Android](#)
- [Azure Storage Client SDK Reference](#)
- [Azure Storage Services REST API](#)
- [Azure Storage Team Blog](#)
- [Transfer data with the AzCopy Command-Line Utility](#)
- [Troubleshooting Azure Files problems - Windows](#)

Develop for Azure Files with C++

7/31/2019 • 8 minutes to read • [Edit Online](#)

TIP

Try the 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.

About this tutorial

In this tutorial, you'll learn how to perform basic operations on Azure Files. Through samples written in C++, you'll learn how to create shares and directories, upload, list, and delete files. If you are new to Azure Files, going through the concepts in the sections that follow will be helpful in understanding the samples.

- Create and delete Azure file shares
- Create and delete directories
- Enumerate files and directories in an Azure file share
- Upload, download, and delete a file
- Set the quota (maximum size) for an Azure file share
- Create a shared access signature (SAS key) for a file that uses a shared access policy defined on the share.

NOTE

Because Azure Files may be accessed over SMB, it is possible to write simple applications that access the Azure file share using the standard C++ I/O classes and functions. This article will describe how to write applications that use the Azure Storage C++ SDK, which uses the [File REST API](#) to talk to Azure Files.

Create a C++ application

To build the samples, you will need to install the Azure Storage Client Library 2.4.0 for C++. You should also have created an Azure storage account.

To install the Azure Storage Client 2.4.0 for C++, you can use one of the following methods:

- **Linux:** Follow the instructions given in the [Azure Storage Client Library for C++ README](#) page.
- **Windows:** In Visual Studio, click **Tools > NuGet Package Manager > Package Manager Console**. Type the following command into the [NuGet Package Manager console](#) and press **ENTER**.

```
Install-Package wastorage
```

Set up your application to use Azure Files

Add the following include statements to the top of the C++ source file where you want to manipulate Azure Files:

```
#include <was/storage_account.h>
#include <was/file.h>
```

Set up an Azure storage connection string

To use File storage, you need to connect to your Azure storage account. The first step would be to configure a connection string, which we'll use to connect to your storage account. Let's define a static variable to do that.

```
// Define the connection-string with your values.
const utility::string_t
storage_connection_string(U("DefaultEndpointsProtocol=https;AccountName=your_storage_account;AccountKey=your_storage_account_key"));
```

Connecting to an Azure 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);
```

Create an Azure file share

All files and directories in an Azure file share reside in a container called a **Share**. Your storage account can have as many shares as your account capacity allows. To obtain access to a share and its contents, you need to use an Azure Files client.

```
// Create the Azure Files client.
azure::storage::cloud_file_client file_client =
    storage_account.create_cloud_file_client();
```

Using the Azure Files client, you can then obtain a reference to a share.

```
// Get a reference to the file share
azure::storage::cloud_file_share share =
    file_client.get_share_reference(_XPLATSTR("my-sample-share"));
```

To create the share, use the **create_if_not_exists** method of the **cloud_file_share** object.

```
if (share.create_if_not_exists()) {
    std::wcout << U("New share created") << std::endl;
}
```

At this point, **share** holds a reference to a share named **my-sample-share**.

Delete an Azure file share

Deleting a share is done by calling the **delete_if_exists** method on a **cloud_file_share** object. Here's sample code that does that.

```

// Get a reference to the share.
azure::storage::cloud_file_share share =
    file_client.get_share_reference(_XPLATSTR("my-sample-share"));

// delete the share if exists
share.delete_share_if_exists();

```

Create a directory

You can organize storage by putting files inside subdirectories instead of having all of them in the root directory. Azure Files allows you to create as many directories as your account will allow. The code below will create a directory named **my-sample-directory** under the root directory as well as a subdirectory named **my-sample-subdirectory**.

```

// Retrieve a reference to a directory
azure::storage::cloud_file_directory directory = share.get_directory_reference(_XPLATSTR("my-sample-
directory"));

// Return value is true if the share did not exist and was successfully created.
directory.create_if_not_exists();

// Create a subdirectory.
azure::storage::cloud_file_directory subdirectory =
    directory.get_subdirectory_reference(_XPLATSTR("my-sample-subdirectory"));
subdirectory.create_if_not_exists();

```

Delete a directory

Deleting a directory is a simple task, although it should be noted that you cannot delete a directory that still contains files or other directories.

```

// Get a reference to the share.
azure::storage::cloud_file_share share =
    file_client.get_share_reference(_XPLATSTR("my-sample-share"));

// Get a reference to the directory.
azure::storage::cloud_file_directory directory =
    share.get_directory_reference(_XPLATSTR("my-sample-directory"));

// Get a reference to the subdirectory you want to delete.
azure::storage::cloud_file_directory sub_directory =
    directory.get_subdirectory_reference(_XPLATSTR("my-sample-subdirectory"));

// Delete the subdirectory and the sample directory.
sub_directory.delete_directory_if_exists();

directory.delete_directory_if_exists();

```

Enumerate files and directories in an Azure file share

Obtaining a list of files and directories within a share is easily done by calling **list_files_and_directories** on a **cloud_file_directory** reference. To access the rich set of properties and methods for a returned **list_file_and_directory_item**, you must call the **list_file_and_directory_item.as_file** method to get a **cloud_file** object, or the **list_file_and_directory_item.as_directory** method to get a **cloud_file_directory** object.

The following code demonstrates how to retrieve and output the URI of each item in the root directory of the

share.

```
//Get a reference to the root directory for the share.  
azure::storage::cloud_file_directory root_dir =  
    share.get_root_directory_reference();  
  
// Output URI of each item.  
azure::storage::list_file_and_directory_result_iterator end_of_results;  
  
for (auto it = directory.list_files_and_directories(); it != end_of_results; ++it)  
{  
    if(it->is_directory())  
    {  
        ucout << "Directory: " << it->as_directory().uri().primary_uri().to_string() << std::endl;  
    }  
    else if (it->is_file())  
    {  
        ucout << "File: " << it->as_file().uri().primary_uri().to_string() << std::endl;  
    }  
}
```

Upload a file

At the very least, an Azure file share contains a root directory where files can reside. In this section, you'll learn how to upload a file from local storage onto the root directory of a share.

The first step in uploading a file is to obtain a reference to the directory where it should reside. You do this by calling the **get_root_directory_reference** method of the share object.

```
//Get a reference to the root directory for the share.  
azure::storage::cloud_file_directory root_dir = share.get_root_directory_reference();
```

Now that you have a reference to the root directory of the share, you can upload a file onto it. This example uploads from a file, from text, and from a stream.

```
// Upload a file from a stream.  
concurrency::streams::istream input_stream =  
    concurrency::streams::file_stream<uint8_t>::open_istream(_XPLATSTR("DataFile.txt")).get();  
  
azure::storage::cloud_file file1 =  
    root_dir.get_file_reference(_XPLATSTR("my-sample-file-1"));  
file1.upload_from_stream(input_stream);  
  
// Upload some files from text.  
azure::storage::cloud_file file2 =  
    root_dir.get_file_reference(_XPLATSTR("my-sample-file-2"));  
file2.upload_text(_XPLATSTR("more text"));  
  
// Upload a file from a file.  
azure::storage::cloud_file file4 =  
    root_dir.get_file_reference(_XPLATSTR("my-sample-file-3"));  
file4.upload_from_file(_XPLATSTR("DataFile.txt"));
```

Download a file

To download files, first retrieve a file reference and then call the **download_to_stream** method to transfer the file contents to a stream object, which you can then persist to a local file. Alternatively, you can use the **download_to_file** method to download the contents of a file to a local file. You can use the **download_text** method to download the contents of a file as a text string.

The following example uses the **download_to_stream** and **download_text** methods to demonstrate downloading the files, which were created in previous sections.

```
// Download as text
azure::storage::cloud_file text_file =
    root_dir.get_file_reference(_XPLATSTR("my-sample-file-2"));
utility::string_t text = text_file.download_text();
ucout << "File Text: " << text << std::endl;

// Download as a stream.
azure::storage::cloud_file stream_file =
    root_dir.get_file_reference(_XPLATSTR("my-sample-file-3"));

concurrency::streams::container_buffer<std::vector<uint8_t>> buffer;
concurrency::streams::ostream output_stream(buffer);
stream_file.download_to_stream(output_stream);
std::ofstream outfile("DownloadFile.txt", std::ofstream::binary);
std::vector<unsigned char>& data = buffer.collection();
outfile.write((char*)&data[0], buffer.size());
outfile.close();
```

Delete a file

Another common Azure Files operation is file deletion. The following code deletes a file named my-sample-file-3 stored under the root directory.

```
// Get a reference to the root directory for the share.
azure::storage::cloud_file_share share =
    file_client.get_share_reference(_XPLATSTR("my-sample-share"));

azure::storage::cloud_file_directory root_dir =
    share.get_root_directory_reference();

azure::storage::cloud_file file =
    root_dir.get_file_reference(_XPLATSTR("my-sample-file-3"));

file.delete_file_if_exists();
```

Set the quota (maximum size) for an Azure file share

You can set the quota (or maximum size) for a file share, in gigabytes. You can also check to see how much data is currently stored on the share.

By setting the quota for a share, you can limit the total size of the files stored on the share. If the total size of files on the share exceeds the quota set on the share, then clients will be unable to increase the size of existing files or create new files, unless those files are empty.

The example below shows how to check the current usage for a share and how to set the quota for the share.

```
// Parse the connection string for the storage account.  
azure::storage::cloud_storage_account storage_account =  
    azure::storage::cloud_storage_account::parse(storage_connection_string);  
  
// Create the file client.  
azure::storage::cloud_file_client file_client =  
    storage_account.create_cloud_file_client();  
  
// Get a reference to the share.  
azure::storage::cloud_file_share share =  
    file_client.get_share_reference(_XPLATSTR("my-sample-share"));  
if (share.exists())  
{  
    std::cout << "Current share usage: " << share.download_share_usage() << "/" << share.properties().quota();  
  
    //This line sets the quota to 2560GB  
    share.resize(2560);  
  
    std::cout << "Quota increased: " << share.download_share_usage() << "/" << share.properties().quota();  
}  
}
```

Generate a shared access signature for a file or file share

You can generate a shared access signature (SAS) for a file share or for an individual file. You can also create a shared access policy on a file share to manage shared access signatures. Creating a shared access policy is recommended, as it provides a means of revoking the SAS if it should be compromised.

The following example creates a shared access policy on a share, and then uses that policy to provide the constraints for a SAS on a file in the share.

```

// Parse the connection string for the storage account.
azure::storage::cloud_storage_account storage_account =
    azure::storage::cloud_storage_account::parse(storage_connection_string);

// Create the file client and get a reference to the share
azure::storage::cloud_file_client file_client =
    storage_account.create_cloud_file_client();

azure::storage::cloud_file_share share =
    file_client.get_share_reference(_XPLATSTR("my-sample-share"));

if (share.exists())
{
    // Create and assign a policy
    utility::string_t policy_name = _XPLATSTR("sampleSharePolicy");

    azure::storage::file_shared_access_policy sharedPolicy =
        azure::storage::file_shared_access_policy();

    //set permissions to expire in 90 minutes
    sharedPolicy.set_expiry(utility::datetime::utc_now() +
        utility::datetime::from_minutes(90));

    //give read and write permissions
    sharedPolicy.set_permissions(azure::storage::file_shared_access_policy::permissions::write |
        azure::storage::file_shared_access_policy::permissions::read);

    //set permissions for the share
    azure::storage::file_share_permissions permissions;

    //retrieve the current list of shared access policies
    azure::storage::shared_access_policies<azure::storage::file_shared_access_policy> policies;

    //add the new shared policy
    policies.insert(std::make_pair(policy_name, sharedPolicy));

    //save the updated policy list
    permissions.set_policies(policies);
    share.upload_permissions(permissions);

    //Retrieve the root directory and file references
    azure::storage::cloud_file_directory root_dir =
        share.get_root_directory_reference();
    azure::storage::cloud_file file =
        root_dir.get_file_reference(_XPLATSTR("my-sample-file-1"));

    // Generate a SAS for a file in the share
    // and associate this access policy with it.
    utility::string_t sas_token = file.get_shared_access_signature(sharedPolicy);

    // Create a new CloudFile object from the SAS, and write some text to the file.
    azure::storage::cloud_file
    file_with_sas(azure::storage::storage_credentials(sas_token).transform_uri(file.uri().primary_uri()));
    utility::string_t text = _XPLATSTR("My sample content");
    file_with_sas.upload_text(text);

    //Download and print URL with SAS.
    utility::string_t downloaded_text = file_with_sas.download_text();
    ucout << downloaded_text << std::endl;
    ucout <<
    azure::storage::storage_credentials(sas_token).transform_uri(file.uri().primary_uri()).to_string() <<
    std::endl;
}

}

```

Next steps

To learn more about Azure Storage, explore these resources:

- [Storage Client Library for C++](#)
- [Azure Storage File Service Samples in C++](#)
- [Azure Storage Explorer](#)
- [Azure Storage Documentation](#)

Develop for Azure Files with Python

7/31/2019 • 4 minutes to read • [Edit Online](#)

TIP

Try the 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.

This tutorial will demonstrate the basics of using Python to develop applications or services that use Azure Files to store file data. In this tutorial, we will create a simple console application and show how to perform basic actions with Python and Azure Files:

- Create Azure file shares
- Create directories
- Enumerate files and directories in an Azure file share
- Upload, download, and delete a file

NOTE

Because Azure Files may be accessed over SMB, it is possible to write simple applications that access the Azure file share using the standard Python I/O classes and functions. This article will describe how to write applications that use the Azure Storage Python SDK, which uses the [Azure Files REST API](#) to talk to Azure Files.

Download and Install Azure Storage SDK for Python

The [Azure Storage SDK for Python](#) requires Python 2.7, 3.3, 3.4, 3.5, or 3.6.

Install via PyPi

To install via the Python Package Index (PyPI), type:

```
pip install azure-storage-file
```

NOTE

If you are upgrading from the Azure Storage SDK for Python version 0.36 or earlier, uninstall the older SDK using `pip uninstall azure-storage` before installing the latest package.

For alternative installation methods, visit the [Azure Storage SDK for Python on GitHub](#).

View the sample application

To view and run a sample application that shows how to use Python with Azure Files, see [Azure Storage: Getting Started with Azure Files in Python](#).

To run the sample application, make sure you have installed both the `azure-storage-file` and

`azure-storage-common` packages.

Set up your application to use Azure Files

Add the following near the top of any Python source file in which you wish to programmatically access Azure Storage.

```
from azure.storage.file import FileService
```

Set up a connection to Azure Files

The `FileService` object lets you work with shares, directories and files. The following code creates a `FileService` object using the storage account name and account key. Replace `<myaccount>` and `<mykey>` with your account name and key.

```
file_service = FileService(account_name='myaccount', account_key='mykey')
```

Create an Azure file share

In the following code example, you can use a `FileService` object to create the share if it doesn't exist.

```
file_service.create_share('myshare')
```

Create a directory

You can also organize storage by putting files inside sub-directories instead of having all of them in the root directory. Azure Files allows you to create as many directories as your account will allow. The code below will create a sub-directory named **sampledir** under the root directory.

```
file_service.create_directory('myshare', 'sampledir')
```

Enumerate files and directories in an Azure file share

To list the files and directories in a share, use the **list_directories_and_files** method. This method returns a generator. The following code outputs the **name** of each file and directory in a share to the console.

```
generator = file_service.list_directories_and_files('myshare')
for file_or_dir in generator:
    print(file_or_dir.name)
```

Upload a file

Azure file share contains at the very least, a root directory where files can reside. In this section, you'll learn how to upload a file from local storage onto the root directory of a share.

To create a file and upload data, use the `create_file_from_path`, `create_file_from_stream`, `create_file_from_bytes` or `create_file_from_text` methods. They are high-level methods that perform the necessary chunking when the size of the data exceeds 64 MB.

`create_file_from_path` uploads the contents of a file from the specified path, and `create_file_from_stream`

uploads the contents from an already opened file/stream. `create_file_from_bytes` uploads an array of bytes, and `create_file_from_text` uploads the specified text value using the specified encoding (defaults to UTF-8).

The following example uploads the contents of the **sunset.png** file into the **myfile** file.

```
from azure.storage.file import ContentSettings
file_service.create_file_from_path(
    'myshare',
    None, # We want to create this blob in the root directory, so we specify None for the directory_name
    'myfile',
    'sunset.png',
    content_settings=ContentSettings(content_type='image/png'))
```

Download a file

To download data from a file, use `get_file_to_path`, `get_file_to_stream`, `get_file_to_bytes`, or `get_file_to_text`. They are high-level methods that perform the necessary chunking when the size of the data exceeds 64 MB.

The following example demonstrates using `get_file_to_path` to download the contents of the **myfile** file and store it to the **out-sunset.png** file.

```
file_service.get_file_to_path('myshare', None, 'myfile', 'out-sunset.png')
```

Delete a file

Finally, to delete a file, call `delete_file`.

```
file_service.delete_file('myshare', None, 'myfile')
```

Create share snapshot

You can create a point in time copy of your entire file share.

```
snapshot = file_service.snapshot_share(share_name)
snapshot_id = snapshot.snapshot
```

Create share snapshot with metadata

```
metadata = {"foo": "bar"}
snapshot = file_service.snapshot_share(share_name, metadata=metadata)
```

List shares and snapshots

You can list all the snapshots for a particular share.

```
shares = list(file_service.list_shares(include_snapshots=True))
```

Browse share snapshot

You can browse content of each share snapshot to retrieve files and directories from that point in time.

```
directories_and_files = list(
    file_service.list_directories_and_files(share_name, snapshot=snapshot_id))
```

Get file from share snapshot

You can download a file from a share snapshot for your restore scenario.

```
with open(FILE_PATH, 'wb') as stream:
    file = file_service.get_file_to_stream(
        share_name, directory_name, file_name, stream, snapshot=snapshot_id)
```

Delete a single share snapshot

You can delete a single share snapshot.

```
file_service.delete_share(share_name, snapshot=snapshot_id)
```

Delete share when share snapshots exist

A share that contains snapshots cannot be deleted unless all the snapshots are deleted first.

```
file_service.delete_share(share_name, delete_snapshots=DeleteSnapshot.Include)
```

Next steps

Now that you've learned how to manipulate Azure Files with Python, follow these links to learn more.

- [Python Developer Center](#)
- [Azure Storage Services REST API](#)
- [Microsoft Azure Storage SDK for Python](#)

Determine which Azure Storage encryption key model is in use for the storage account

1/5/2020 • 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

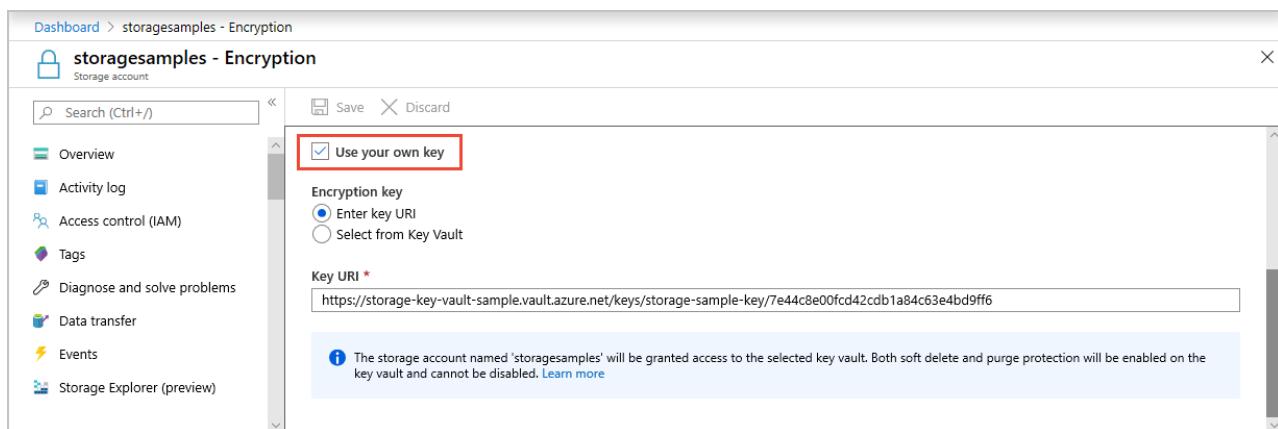
To determine whether a storage account is using Microsoft-managed keys or customer-managed keys for encryption, use one of the following approaches.

- [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 where customer-managed keys are in use for encryption:



Next steps

[Azure Storage encryption for data at rest](#)

Configure customer-managed keys with Azure Key Vault by using the Azure portal

1/5/2020 • 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:

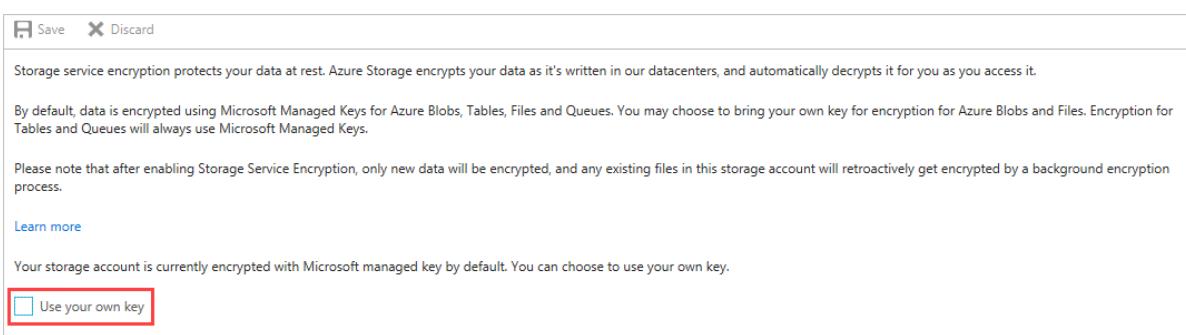
- [How to use soft-delete with PowerShell](#).
- [How to use soft-delete with CLI](#).

Only RSA keys of size 2048 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 **Use your own key** option, as shown in the following figure.



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 next to it. Under 'Settings', the 'Enabled?' switch is set to 'Yes'. The 'Permitted operations' section includes checked boxes for Encrypt, Sign, Wrap Key, Decrypt, Verify, and Unwrap Key. A 'Tags' section shows 0 tags.

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 'Encryption key' section has the 'Enter key URI' radio button selected. Below it, 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' settings page for an Azure storage account. At the top, there are 'Save' and 'Discard' buttons. Below them is a note about storage service encryption protecting data at rest. A section titled 'Encryption key' has a checked checkbox for 'Use your own key'. Underneath, there are two options: 'Enter key URI' (radio button) and 'Select from Key Vault' (radio button, which is selected). A detailed view of the 'Select from Key Vault' section shows a list with an item 'Key Vault <key-vault>' and an 'Encryption key <key>' item, both preceded by an asterisk indicating they are required. A note at the bottom states that the storage account will be granted access to the selected key vault, enabling soft delete and purge protection.

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 then 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

1/5/2020 • 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 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,recover
```

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'
```

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.

Disable customer-managed keys

When you disable customer-managed keys, your storage account is then 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

1/13/2020 • 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 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 recover 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>
```

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.

Disable customer-managed keys

When you disable customer-managed keys, your storage account is then 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?](#)

Configure Azure Storage firewalls and virtual networks

1/22/2020 • 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 inbound 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). Learn more .
Azure Data Box	Microsoft.DataBox	Enables import of data to Azure using Data Box. Learn more .
Azure DevTest Labs	Microsoft.DevTestLab	Custom image creation and artifact installation. Learn more .
Azure Event Grid	Microsoft.EventGrid	Enable Blob Storage event publishing and allow Event Grid to publish to storage queues. Learn about blob storage events and publishing to queues .
Azure Event Hubs	Microsoft.EventHub	Archive data with Event Hubs Capture. Learn More .
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. Learn more
Azure HDInsight	Microsoft.HDInsight	Provision the initial contents of the default file system for a new HDInsight cluster. Learn more .
Azure Import Export	Microsoft.ImportExport	Enables import of data to Azure and export of data from Azure using Import/Export service. Learn more .
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. Learn more .
Azure Networking	Microsoft.Network	Store and analyze network traffic logs. Learn more .
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. Learn more .

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 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 Logic Apps	Microsoft.Logic/workflows	Enables logic apps to access storage accounts. Learn more .
Azure Machine Learning	Microsoft.MachineLearningServices	Authorized Azure Machine Learning workspaces write experiment output, models, and logs to Blob storage. Learn more .
Azure SQL Data Warehouse	Microsoft.Sql	Allows import and export of data from specific SQL Database instances using PolyBase. Learn more .
Azure Stream Analytics	Microsoft.StreamAnalytics	Allows data from a streaming job to be written to Blob storage. This feature is currently in preview. Learn more .
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.

CL Iv2

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](#).

Require secure transfer to ensure secure connections

12/13/2019 • 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 in Azure portal. However, it is disabled when you create a storage account with SDK.

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 Disabled

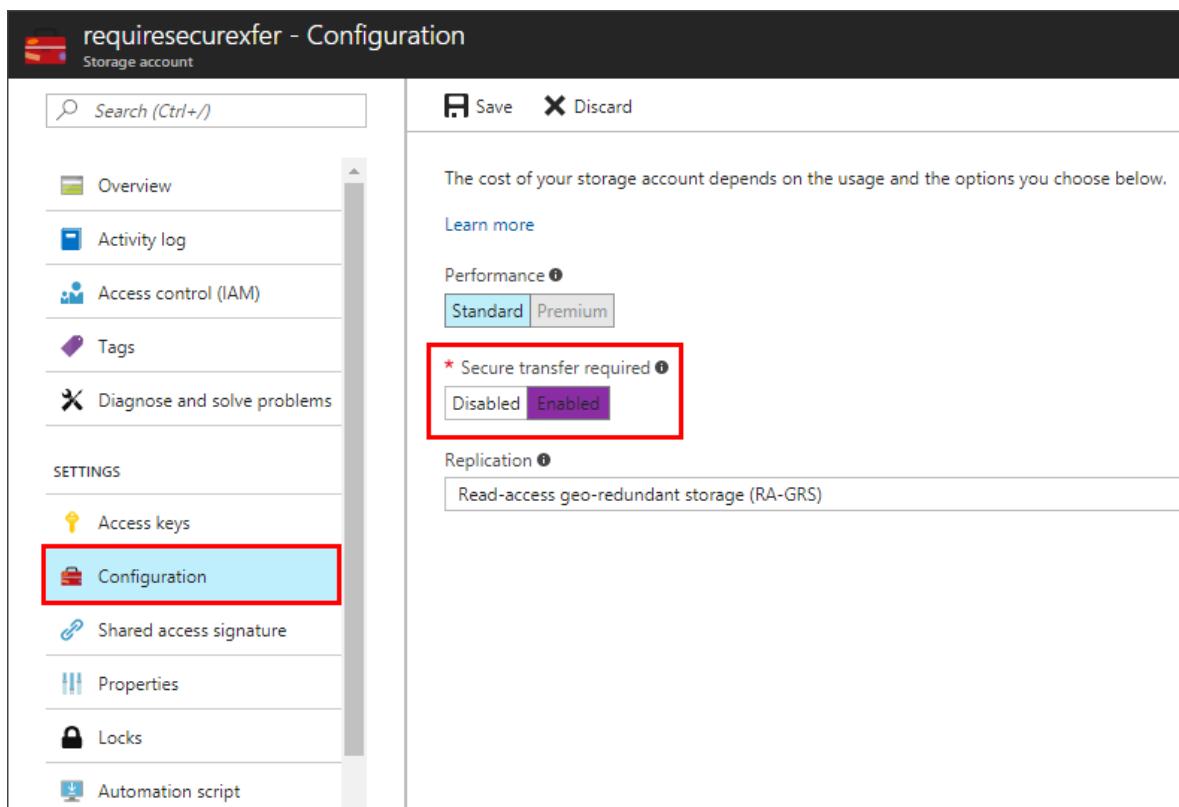
VIRTUAL NETWORKS

Allow access from All networks Selected network
DATA LAKE STORAGE GEN2 (PREVIEW)
Hierarchical namespace 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**.



The screenshot shows the Azure Storage account configuration page for a storage account named "requiresecurexfer". The left sidebar lists various settings: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, SETTINGS (Access keys, Configuration, Shared access signature, Properties, Locks, Automation script), and Replication (Read-access geo-redundant storage (RA-GRS)). The "Configuration" item is highlighted with a red box. The main pane shows the "Configuration" tab with options for Performance (Standard selected, Premium available), Secure transfer required (Enabled selected, Disabled available), and Replication (RA-GRS selected). A note states: "The cost of your storage account depends on the usage and the options you choose below." A "Learn more" link is also present.

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](#)

Enable Active Directory authentication over SMB for Azure file shares

2/28/2020 • 16 minutes to read • [Edit Online](#)

Azure File supports identity-based authentication over Server Message Block (SMB) through two types of Domain Services: Azure Active Directory Domain Services (Azure AD DS) (GA) and Active Directory (AD) (preview). This article focuses on the newly introduced (preview) support of leveraging Active Directory Domain Service for authentication to Azure file shares. If you are interested in enabling Azure AD DS (GA) authentication for Azure file shares, refer to [our article on the subject](#).

NOTE

Azure file shares only support authentication against one domain service, either Azure Active Directory Domain Service (Azure AD DS) or Active Directory (AD).

AD identities used for Azure file share authentication must be synced to Azure AD. Password hash synchronization is optional.

AD authentication does not support authentication against Computer accounts created in AD.

AD authentication can only be supported against one AD forest where the storage account is registered to. You can only access Azure file shares with the AD credentials from a single AD forest by default. If you need to access your Azure file share from a different forest, make sure that you have the proper forest trust configured, see [FAQ](#) for details.

AD authentication for SMB access and ACL persistence is supported for Azure file shares managed by Azure File Sync.

Azure Files supports Kerberos authentication with AD with RC4-HMAC encryption. AES Kerberos encryption is not yet supported.

When you enable AD for Azure file shares over SMB, your AD domain joined machines can mount Azure file shares using your existing AD credentials. This capability can be enabled with an AD environment hosted either in on-prem machines or hosted in Azure.

AD identities used to access Azure file shares must be synced to Azure AD to enforce share level file permissions through the standard [role-based access control \(RBAC\)](#) model. [Windows-style DACLs](#) on files/directories carried over from existing file servers will be preserved and enforced. This feature offers seamless integration with your enterprise AD domain infrastructure. As you replace on-prem file servers with Azure file shares, existing users can access Azure file shares from their current clients with a single sign-on experience, without any change to the credentials in use.

Prerequisites

Before you enable AD authentication for Azure file shares, make sure you have completed the following prerequisites:

- Select or create your AD environment and sync it to Azure AD.

You can enable the feature on a new or existing AD environment. Identities used for access must be synced to Azure AD. The Azure AD tenant and the file share that you are accessing must be associated with the same subscription.

To setup an AD domain environment, refer to [Active Directory Domain Services Overview](#). If you have not synced your AD to your Azure AD, follow the guidance in [What is hybrid identity with Azure Active](#)

[Directory?](#) in order to determine your preferred authentication method and Azure AD Connect setup option.

- Domain-join an on-premises machine or an Azure VM to AD (also referred as AD DS).

To access a file share by using AD credentials from a machine or VM, your device must be domain-joined to AD. For information about how to domain-join to AD, refer to [Join a Computer to a Domain](#).

- Select or create an Azure storage account in [a supported region](#).

Make sure that the storage account containing your file shares is not already configured for Azure AD DS Authentication. If Azure Files Azure AD DS Authentication is enabled on the storage account, it needs to be disabled before changing to use AD. This implies that existing ACLs configured in Azure AD DS environment will need to be reconfigured for proper permission enforcement.

For information about creating a new file share, see [Create a file share in Azure Files](#).

For optimal performance, we recommend that you deploy the storage account in the same region as the VM from which you plan to access the share.

- Verify connectivity by mounting Azure file shares using your storage account key.

To verify that your device and file share are properly configured, try mounting the file share using your storage account key. For more information, see [Use an Azure file share with Windows](#).

Regional availability

Azure Files AD authentication (preview) is available in [most public regions](#) except for:

- West US
- West US 2
- East US
- East US 2
- South Central US
- West Europe
- North Europe

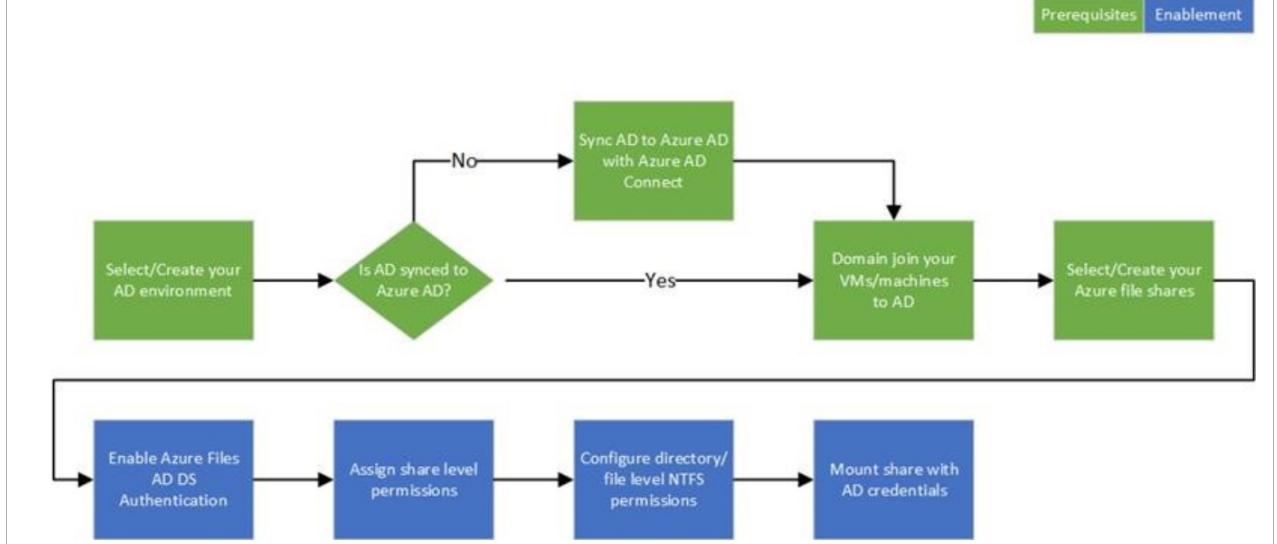
Workflow overview

Before you enable AD Authentication over SMB for Azure file shares, we recommend that you walk through the [prerequisites](#) and make sure you've completed all the steps. The prerequisites validate that your AD, Azure AD, and Azure Storage environments are properly configured.

Next, grant access to Azure Files resources with AD credentials:

- Enable Azure Files AD authentication on your storage account.
- Assign access permissions for a share to the Azure AD identity (a user, group, or service principal) that is in sync with the target AD identity.
- Configure ACLs over SMB for directories and files.
- Mount an Azure file share from an AD domain joined VM.

The following diagram illustrates the end-to-end workflow for enabling Azure AD authentication over SMB for Azure file shares.

**NOTE**

AD authentication over SMB for Azure file shares is only supported on machines or VMs running on OS versions newer than Windows 7 or Windows Server 2008 R2.

Enable AD authentication for your account

To enable AD authentication over SMB for Azure file shares, you need to first register your storage account with AD and then set the required domain properties on the storage account. When the feature is enabled on the storage account, it applies to all new and existing file shares in the account. Use `join-AzStorageAccountForAuth` to enable the feature. You can find the detailed description of the end-to-end workflow in the section below.

IMPORTANT

The `join-AzStorageAccountForAuth` cmdlet will make modifications to your AD environment. Read the following explanation to better understand what it is doing to ensure you have the proper permissions to execute the command and that the applied changes align with the compliance and security policies.

The `join-AzStorageAccountForAuth` cmdlet will perform the equivalent of an offline domain join on behalf of the indicated storage account. It will create an account in your AD domain, either a [computer account](#) or a [service logon account](#). The created AD account represents the storage account in the AD domain. If the AD account is created under an AD Organizational Unit (OU) that enforces password expiration, you must update the password before the maximum password age. Failing to update AD account password will result in authentication failures when accessing Azure file shares. To learn how to update the password, see [Update AD account password](#).

You can use the following script to perform the registration and enable the feature or, alternatively, you can manually perform the operations that the script would. Those operations are described in the section following the script. You do not need to do both.

1. Check prerequisites

- [Download and unzip the AzFilesHybrid module](#)
- Install and execute the module in a device that is domain joined to AD with AD credentials that have permissions to create a service logon account or a computer account in the target AD.
- Run the script using an AD credential that is synced to your Azure AD. The AD credential must have either the storage account owner or the contributor RBAC role permissions.
- Make sure your storage account is in a [supported region](#).

2. Execute AD enablement script

Remember to replace the placeholder values with your own in the parameters below before executing it in PowerShell.

```
#Change the execution policy to unblock importing AzFilesHybrid.psm1 module
Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Scope CurrentUser

# Navigate to where AzFilesHybrid is unzipped and stored and run to copy the files into your path
.\CopyToPSPPath.ps1

#Import AzFilesHybrid module
Import-Module -name AzFilesHybrid

#Login with an Azure AD credential that has either storage account owner or contributer RBAC assignment
Connect-AzAccount

#Select the target subscription for the current session
Select-AzSubscription -SubscriptionId "<your-subscription-id-here>"

#Register the target storage account with your active directory environment under the target OU
join-AzStorageAccountForAuth -ResourceGroupName "<resource-group-name-here>" -Name "<storage-account-name-here>" -DomainAccountType "<ServiceLogonAccount|ComputerAccount>" -OrganizationalUnitName "<ou-name-here>"
```

The following description summarizes all actions performed when the `join-AzStorageAccountForAuth` cmdlet gets executed. You may perform these steps manually, if you prefer not to use the command:

NOTE

If you have already executed the `join-AzStorageAccountForAuth` script above successfully, go to the next section "3. Confirm that the feature is enabled". You do not need to perform the operations below again.

a. Checking environment

First, it checks your environment. Specifically it checks if the [Active Directory PowerShell](#) is installed and if the shell is being executed with administrator privileges. Then it checks to see if the [Az.Storage 1.11.1-preview module](#) is installed, and installs it if it isn't. If those checks pass, then it will check your AD to see if there is either a [computer account](#) (default) or [service logon account](#) that has already been created with SPN/UPN as "cifs/your-storage-account-name-here.file.core.windows.net". If the account doesn't exist, it will create one as described in section b below.

b. Creating an identity representing the storage account in your AD manually

To create this account manually, create a new kerberos key for your storage account using

`New-AzStorageAccountKey -KeyName kerb1`. Then, use that kerberos key as the password for your account. This key is only used during set up and cannot be used for any control or data plane operations against the storage account.

Once you have that key, create either a service or computer account under your OU. Use the following specification: SPN: "cifs/your-storage-account-name-here.file.core.windows.net" Password: Kerberos key for your storage account.

If your OU enforces password expiration, you must update the password before the maximum password age to prevent authentication failures when accessing Azure file shares. See [Update AD account password](#) for details.

Keep the SID of the newly created account, you'll need it for the next step.

c. Enable the feature on your storage account

The script would then enable the feature on your storage account. To perform this setup manually, provide some configuration details for the domain properties in the following command, then run it. The storage account SID required in the following command is the SID of the identity you created in AD (section b above).

```
#Set the feature flag on the target storage account and provide the required AD domain information

Set-AzStorageAccount -ResourceGroupName "<your-resource-group-name-here>" -Name "<your-storage-account-name-here>" -EnableActiveDirectoryDomainServicesForFile $true -ActiveDirectoryDomainName "<your-domain-name-here>" -ActiveDirectoryNetBiosDomainName "<your-netbios-domain-name-here>" -ActiveDirectoryForestName "<your-forest-name-here>" -ActiveDirectoryDomainGuid "<your-guid-here>" -ActiveDirectoryDomainsid "<your-domain-sid-here>" -ActiveDirectoryAzureStorageSid "<your-storage-account-sid>"
```

3. Confirm that the feature is enabled

You can check to confirm whether the feature is enabled on your storage account, you can use the following script:

```
#Get the target storage account
$storageaccount = Get-AzStorageAccount -ResourceGroupName "<your-resource-group-name-here>" -Name "<your-storage-account-name-here>"

#List the directory service of the selected service account
$storageAccount.AzureFilesIdentityBasedAuth.DirectoryServiceOptions

#List the directory domain information if the storage account has enabled AD authentication for file shares
$storageAccount.AzureFilesIdentityBasedAuth.ActiveDirectoryProperties
```

You've now successfully enabled the feature on your storage account. Even though the feature is enabled, you still need to complete additional actions to be able to use the feature properly.

Assign access permissions to an identity

To access Azure Files resources with identity based authentication, an identity (a user, group, or service principal) must have the necessary permissions at the share level. This process is similar to specifying Windows share permissions, where you specify the type of access that a particular user has to a file share. The guidance in this section demonstrates how to assign read, write, or delete permissions for a file share to an identity.

We have introduced three Azure built-in roles for granting share-level permissions to users:

- **Storage File Data SMB Share Reader** allows read access in Azure Storage file shares over SMB.
- **Storage File Data SMB Share Contributor** allows read, write, and delete access in Azure Storage file shares over SMB.
- **Storage File Data SMB Share Elevated Contributor** allows read, write, delete and modify NTFS permissions in Azure Storage file shares over SMB.

IMPORTANT

Full administrative control of a file share, including the ability to assign a role to an identity, requires using the storage account key. Administrative control is not supported with Azure AD credentials.

You can use the Azure portal, PowerShell, or Azure CLI to assign the built-in roles to the Azure AD identity of a user for granting share-level permissions.

NOTE

Remember to sync your AD credentials to Azure AD if you plan to use your AD for authentication. Password hash sync from AD to Azure AD is optional. Share level permission will be granted to the Azure AD identity that is synced from AD.

Azure portal

To assign an RBAC role to an Azure AD identity, using the [Azure portal](#), follow these steps:

1. In the Azure portal, go to your file share, or [Create a file share](#).
2. Select **Access Control (IAM)**.
3. Select **Add a role assignment**
4. In the **Add role assignment** blade, select the appropriate built-in role (Storage File Data SMB Share Reader, Storage File Data SMB Share Contributor) from the **Role** list. Leave **Assign access to** at the default setting: **Azure AD user, group, or service principal**. Select the target Azure AD identity by name or email address.
5. Select **Save** to complete the role assignment operation.

PowerShell

The following PowerShell sample shows how to assign an RBAC role to an Azure AD identity, based on sign-in name. For more information about assigning RBAC roles with PowerShell, see [Manage access using RBAC and Azure PowerShell](#).

Before you run the following sample script, remember to replace placeholder values, including brackets, with your own values.

```
#Get the name of the custom role
$FileShareContributorRole = Get-AzRoleDefinition "<role-name>" #Use one of the built-in roles: Storage File Data SMB Share Reader, Storage File Data SMB Share Contributor, Storage File Data SMB Share Elevated Contributor
#Constrain the scope to the target file share
$scope = "/subscriptions/<subscription-id>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>/fileServices/default/fileshares/<share-name>"
#Assign the custom role to the target identity with the specified scope.
New-AzRoleAssignment -SignInName <user-principal-name> -RoleDefinitionName $FileShareContributorRole.Name -Scope $scope
```

CLI

The following CLI 2.0 command shows how to assign an RBAC role to an Azure AD identity, based on sign-in name. For more information about assigning RBAC roles with Azure CLI, see [Manage access by using RBAC and Azure CLI](#).

Before you run the following sample script, remember to replace placeholder values, including brackets, with your own values.

```
#Assign the built-in role to the target identity: Storage File Data SMB Share Reader, Storage File Data SMB Share Contributor, Storage File Data SMB Share Elevated Contributor
az role assignment create --role "<role-name>" --assignee <user-principal-name> --scope
"/subscriptions/<subscription-id>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>/fileServices/default/fileshares/<share-name>"
```

Configure NTFS permissions over SMB

After you assign share-level permissions with RBAC, you must assign proper NTFS permissions at the root, directory, or file level. Think of share-level permissions as the high-level gatekeeper that determines whether a user can access the share. Whereas NTFS permissions act at a more granular level to determine what operations the user can do at the directory or file level.

Azure Files supports the full set of NTFS basic and advanced permissions. You can view and configure NTFS permissions on directories and files in an Azure file share by mounting the share and then using Windows File Explorer or running the Windows [icacls](#) or [Set-ACL](#) command.

To configure NTFS with superuser permissions, you must mount the share by using your storage account key from your domain-joined VM. Follow the instructions in the next section to mount an Azure file share from the

command prompt and to configure NTFS permissions accordingly.

The following sets of permissions are supported on the root directory of a file share:

- BUILTIN\Administrators:(OI)(CI)(F)
- NT AUTHORITY\SYSTEM:(OI)(CI)(F)
- BUILTIN\Users:(RX)
- BUILTIN\Users:(OI)(CI)(IO)(GR,GE)
- NT AUTHORITY\Authenticated Users:(OI)(CI)(M)
- NT AUTHORITY\SYSTEM:(F)
- CREATOR OWNER:(OI)(CI)(IO)(F)

Configure NTFS permissions with icacls

Use the following Windows command to grant full permissions to all directories and files under the file share, including the root directory. Remember to replace the placeholder values in the example with your own values.

```
icacls <mounted-drive-letter>: /grant <user-email>:(f)
```

For more information on how to use icacls to set NTFS permissions and on the different types of supported permissions, see [the command-line reference for icacls](#).

Mount a file share from the command prompt

Use the Windows **net use** command to mount the Azure file share. Remember to replace the placeholder values in the following example with your own values. For more information about mounting file shares, see [Use an Azure file share with Windows](#).

```
net use <desired-drive-letter>: \\<storage-account-name>.file.core.windows.net\<share-name> <storage-account-key> /user:Azure\<storage-account-name>
```

Configure NTFS permissions with Windows File Explorer

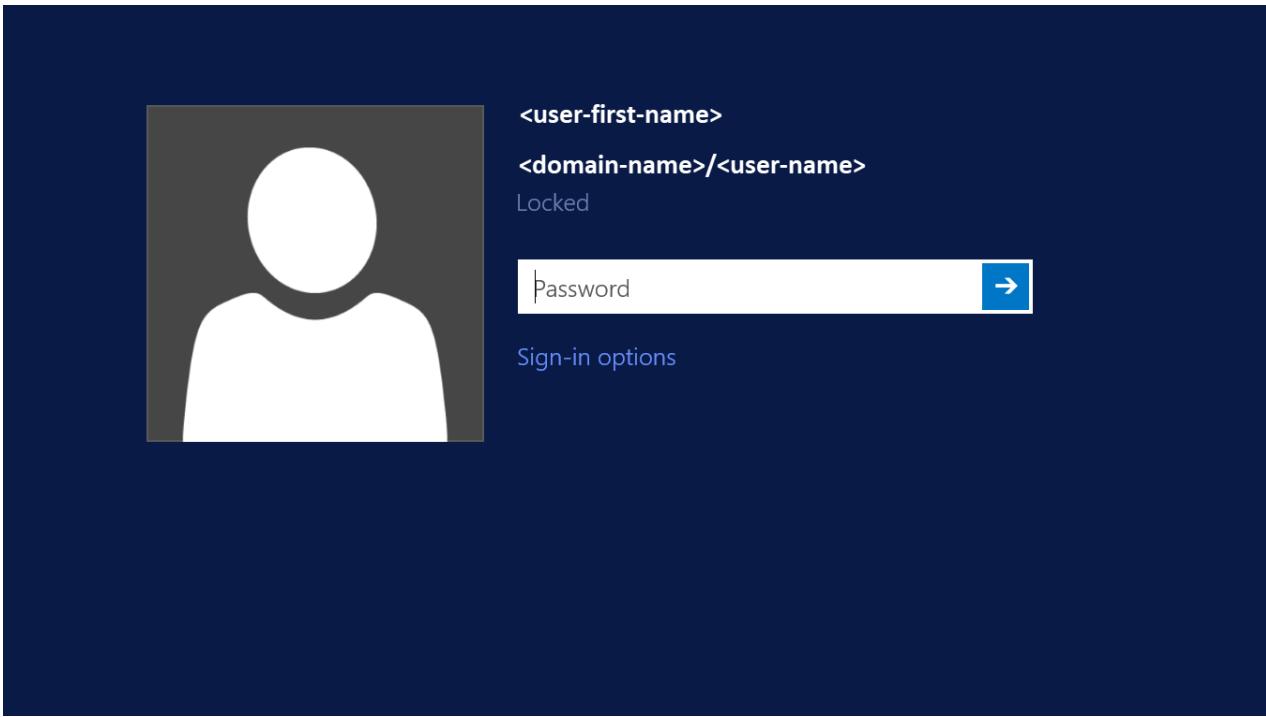
Use Windows File Explorer to grant full permission to all directories and files under the file share, including the root directory.

1. Open Windows File Explorer and right click on the file/directory and select **Properties**
2. Click on the **Security** tab
3. Click on **Edit...** button to change permissions
4. You can change the permission of existing users, or click on **Add...** to grant permissions to new users
5. In the prompt window for adding new users, enter the target user name you want to grant permission to in the **Enter the object names to select** box, and click on **Check Names** to find the full UPN name of the target user.
6. Click on **OK**
7. In the Security tab, select all permissions you want to grant to the newly add user
8. Click on **Apply**

Mount a file share from a domain-joined VM

The following process verifies that your file share and access permissions were set up correctly and that you can access an Azure File share from a domain-joined VM:

Sign in to the VM by using the Azure AD identity to which you have granted permissions, as shown in the following image. If you have enabled AD authentication for Azure Files, use the AD credential. For Azure AD DS authentication, log in with Azure AD credential.



Use the following command to mount the Azure file share. Remember to replace the placeholder values with your own values. Because you've been authenticated, you don't need to provide the storage account key, the AD credentials, or the Azure AD credentials. Single sign-on experience is supported for authentication with either AD or Azure AD DS.

```
net use <desired-drive-letter>: \\<storage-account-name>.file.core.windows.net\<share-name>
```

You have now successfully enabled AD authentication over SMB and assigned a custom role that provides access to an Azure file share with an AD identity. To grant additional users access to your file share, follow the instructions in the [Assign access permissions](#) to use an identity and [Configure NTFS permissions over SMB](#) sections.

Update AD account password

If you registered the AD identity/account representing your storage account under an OU that enforces password expiration time, you must rotate the password before the maximum password age. Failing to update the password of the AD account will result in authentication failures to access Azure file shares.

To trigger password rotation, you can run the `Update-AzStorageAccountADObjectPassword` command from the AzFilesHybrid module. The cmdlet performs actions similar to storage account key rotation. It gets the second Kerberos key of the storage account and uses it to update the password of the registered account in AD. Then it regenerates the target Kerberos key of the storage account and updates the password of the registered account in AD. You must run this cmdlet in an AD domain joined environment.

```
#Update the password of the AD account registered for the storage account
Update-AzStorageAccountADObjectPassword -RotateToKerbKey kerb2 -ResourceGroupName "your-resource-group-name-here" -StorageAccountName "your-storage-account-name-here"
```

Next steps

For more information about Azure Files and how to use AD over SMB, see these resources:

- [Overview of Azure Files identity-based authentication support for SMB access](#)
- [FAQ](#)

Enable Azure Active Directory Domain Services authentication on Azure Files

2/25/2020 • 12 minutes to read • [Edit Online](#)

Azure Files supports identity-based authentication over Server Message Block (SMB) through [Active Directory \(AD\)](#) (preview) and [Azure Active Directory Domain Services \(Azure AD DS\)](#) (GA). This article focuses on how Azure Files can leverage domain services, either on-premises or in Azure, to support identity-based access to Azure Files over SMB. This allows you to easily replace your existing file servers with Azure Files and continue to use your existing directory service, maintaining seamless user access to shares.

Azure Files enforces authorization on the user access to both the share and the directory/file level. Share-level permission assignment can be assigned to Azure AD users or groups managed through the typical [role-based access control \(RBAC\)](#) model. With RBAC, the credentials you use for file access should be available or synced to Azure AD. You can assign built-in RBAC roles like Storage File Data SMB Share Reader to users or groups in Azure AD to grant read access to an Azure file share.

At the directory/file level, Azure Files supports preserving, inheriting, and enforcing [Windows DACLs](#) just like any Windows file servers. If you copy data over SMB from a file share to Azure Files, or vice versa, you can choose to keep Windows DACLs. Whether you plan to enforce authorization or not, you can leverage Azure Files to backup ACLs along with your data.

For an overview of Azure AD authentication over SMB for Azure file shares, see [Overview of Azure Active Directory authentication over SMB for Azure Files](#). This article is focused on how to enable authentication with Azure Active Directory Domain Services (Azure AD DS) on Azure Files.

NOTE

Azure Files supports Kerberos authentication with Azure AD DS with RC4-HMAC encryption. AES Kerberos encryption is not yet supported.

Prerequisites

Before you enable Azure AD over SMB for Azure file shares, make sure you have completed the following prerequisites:

1. Select or create an Azure AD tenant.

You can use a new or existing tenant for Azure AD authentication over SMB. The tenant and the file share that you want to access must be associated with the same subscription.

To create a new Azure AD tenant, you can [Add an Azure AD tenant and an Azure AD subscription](#). If you have an existing Azure AD tenant but want to create a new tenant for use with Azure file shares, see [Create an Azure Active Directory tenant](#).

2. Enable Azure AD Domain Services on the Azure AD tenant.

To support authentication with Azure AD credentials, you must enable Azure AD Domain Services for your Azure AD tenant. If you aren't the administrator of the Azure AD tenant, contact the administrator and follow the step-by-step guidance to [Enable Azure Active Directory Domain Services using the Azure portal](#).

It typically takes about 15 minutes for an Azure AD DS deployment to complete. Verify that the health

status of Azure AD DS shows **Running**, with password hash synchronization enabled, before proceeding to the next step.

3. Domain-join an Azure VM with Azure AD DS.

To access a file share by using Azure AD credentials from a VM, your VM must be domain-joined to Azure AD DS. For more information about how to domain-join a VM, see [Join a Windows Server virtual machine to a managed domain](#).

NOTE

Azure AD DS authentication over SMB with Azure file shares is supported only on Azure VMs running on OS versions above Windows 7 or Windows Server 2008 R2.

4. Select or create an Azure file share.

Select a new or existing file share that's associated with the same subscription as your Azure AD tenant. For information about creating a new file share, see [Create a file share in Azure Files](#). For optimal performance, we recommend that your file share be in the same region as the VM from which you plan to access the share.

5. Verify Azure Files connectivity by mounting Azure file shares using your storage account key.

To verify that your VM and file share are properly configured, try mounting the file share using your storage account key. For more information, see [Mount an Azure file share and access the share in Windows](#).

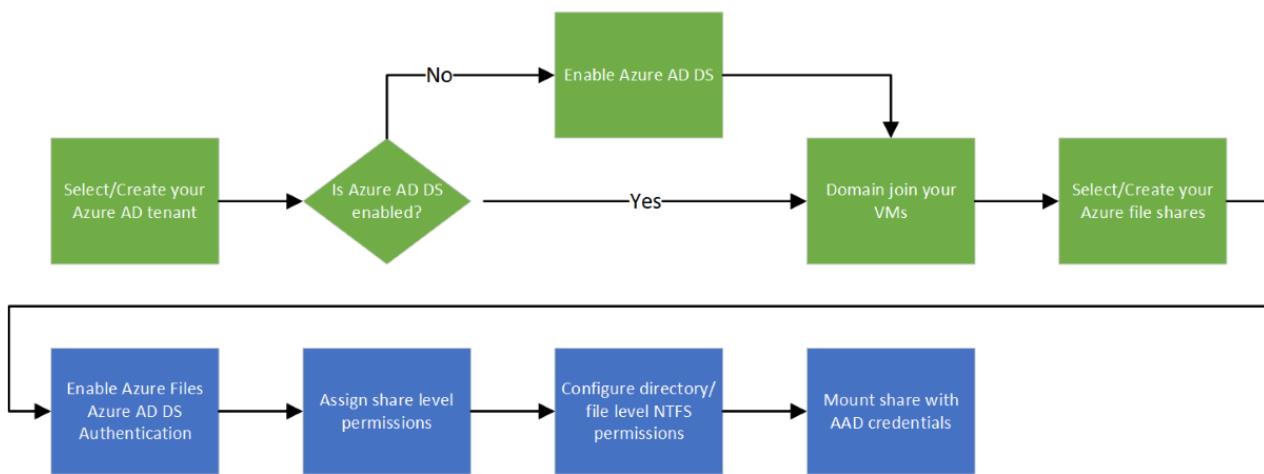
Overview of the workflow

Before you enable Azure AD DS Authentication over SMB for Azure file shares, verify that your Azure AD and Azure Storage environments are properly configured. We recommend that you walk through the [prerequisites](#) to make sure you've completed all the required steps.

Next, do the following things to grant access to Azure Files resources with Azure AD credentials:

- Enable Azure AD DS authentication over SMB for your storage account to register the storage account with the associated Azure AD DS deployment.
- Assign access permissions for a share to an Azure AD identity (a user, group, or service principal).
- Configure NTFS permissions over SMB for directories and files.
- Mount an Azure file share from a domain-joined VM.

The following diagram illustrates the end-to-end workflow for enabling Azure AD DS authentication over SMB for Azure Files.



Enable Azure AD DS authentication for your account

To enable Azure AD DS authentication over SMB for Azure Files, you can set a property on storage accounts by using the Azure portal, Azure PowerShell, or Azure CLI. Setting this property implicitly "domain joins" the storage account with the associated Azure AD DS deployment. Azure AD DS authentication over SMB is then enabled for all new and existing file shares in the storage account.

Keep in mind that you can enable Azure AD DS authentication over SMB only after you have successfully deployed Azure AD DS to your Azure AD tenant. For more information, see the [prerequisites](#).

Azure portal

To enable Azure AD DS authentication over SMB with the [Azure portal](#), follow these steps:

1. In the Azure portal, go to your existing storage account, or [create a storage account](#).
2. In the **Settings** section, select **Configuration**.
3. Under **Identity-based access for file shares** switch the toggle for **Azure Active Directory Domain Service (AAD DS) to Enabled**.
4. Select **Save**.

The following image shows how to enable Azure AD DS authentication over SMB for your storage account.

The screenshot shows the Azure Storage account configuration page for 'myexampleaccountname'. The 'Configuration' tab is selected. Key settings shown include:

- Secure transfer required: Enabled
- Access tier (default): Hot
- Replication: Locally-redundant storage (LRS)
- Large file shares: Enabled
- Azure Active Directory Domain Service (AAD DS): Enabled
- Data Lake Storage Gen2: Enabled
- Hierarchical namespace: Enabled

PowerShell

To enable Azure AD DS authentication over SMB with Azure PowerShell, install the latest Az module (2.4 or newer) or the Az.Storage module (1.5 or newer). For more information about installing PowerShell, see [Install Azure PowerShell on Windows with PowerShellGet](#).

To create a new storage account, call `New-AzStorageAccount`, and then set the `EnableAzureActiveDirectoryDomainServicesForFile` parameter to `true`. In the following example, remember to replace the placeholder values with your own values. (If you were using the previous preview module, the parameter for feature enablement is `EnableAzureFilesAadIntegrationForSMB`.)

```
# Create a new storage account
New-AzStorageAccount -ResourceGroupName "<resource-group-name>" ` 
    -Name "<storage-account-name>" ` 
    -Location "<azure-region>" ` 
    -SkuName Standard_LRS ` 
    -Kind StorageV2 ` 
    -EnableAzureActiveDirectoryDomainServicesForFile $true
```

To enable this feature on existing storage accounts, use the following command:

```
# Update a storage account
Set-AzStorageAccount -ResourceGroupName "<resource-group-name>" ` 
    -Name "<storage-account-name>" ` 
    -EnableAzureActiveDirectoryDomainServicesForFile $true
```

Azure CLI

To enable Azure AD authentication over SMB with Azure CLI, install the latest CLI version (Version 2.0.70 or newer). For more information about installing Azure CLI, see [Install the Azure CLI](#).

To create a new storage account, call `az storage account create`, and set the `--enable-files-aadds` property to `true`. In the following example, remember to replace the placeholder values with your own values. (If you were using the previous preview module, the parameter for feature enablement is `file-aad`.)

```
# Create a new storage account
az storage account create -n <storage-account-name> -g <resource-group-name> --enable-files-aadds $true
```

To enable this feature on existing storage accounts, use the following command:

```
# Update a new storage account
az storage account update -n <storage-account-name> -g <resource-group-name> --enable-files-aadds $true
```

Assign access permissions to an identity

To access Azure Files resources with identity based authentication, an identity (a user, group, or service principal) must have the necessary permissions at the share level. This process is similar to specifying Windows share permissions, where you specify the type of access that a particular user has to a file share. The guidance in this section demonstrates how to assign read, write, or delete permissions for a file share to an identity.

We have introduced three Azure built-in roles for granting share-level permissions to users:

- **Storage File Data SMB Share Reader** allows read access in Azure Storage file shares over SMB.
- **Storage File Data SMB Share Contributor** allows read, write, and delete access in Azure Storage file shares over SMB.
- **Storage File Data SMB Share Elevated Contributor** allows read, write, delete and modify NTFS permissions in Azure Storage file shares over SMB.

IMPORTANT

Full administrative control of a file share, including the ability to assign a role to an identity, requires using the storage account key. Administrative control is not supported with Azure AD credentials.

You can use the Azure portal, PowerShell, or Azure CLI to assign the built-in roles to the Azure AD identity of a user for granting share-level permissions.

NOTE

Remember to sync your AD credentials to Azure AD if you plan to use your AD for authentication. Password hash sync from AD to Azure AD is optional. Share level permission will be granted to the Azure AD identity that is synced from AD.

Azure portal

To assign an RBAC role to an Azure AD identity, using the [Azure portal](#), follow these steps:

1. In the Azure portal, go to your file share, or [Create a file share](#).
2. Select **Access Control (IAM)**.
3. Select **Add a role assignment**
4. In the **Add role assignment** blade, select the appropriate built-in role (Storage File Data SMB Share Reader, Storage File Data SMB Share Contributor) from the **Role** list. Leave **Assign access to** at the default setting: **Azure AD user, group, or service principal**. Select the target Azure AD identity by name or email address.
5. Select **Save** to complete the role assignment operation.

PowerShell

The following PowerShell sample shows how to assign an RBAC role to an Azure AD identity, based on sign-in name. For more information about assigning RBAC roles with PowerShell, see [Manage access using RBAC and Azure PowerShell](#).

Before you run the following sample script, remember to replace placeholder values, including brackets, with your own values.

```
#Get the name of the custom role
$FileShareContributorRole = Get-AzRoleDefinition "<role-name>" #Use one of the built-in roles: Storage File Data SMB Share Reader, Storage File Data SMB Share Contributor, Storage File Data SMB Share Elevated Contributor
#Constrain the scope to the target file share
$scope = "/subscriptions/<subscription-id>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>/fileServices/default/fileshares/<share-name>"
#Assign the custom role to the target identity with the specified scope.
New-AzRoleAssignment -SignInName <user-principal-name> -RoleDefinitionName $FileShareContributorRole.Name -Scope $scope
```

CLI

The following CLI 2.0 command shows how to assign an RBAC role to an Azure AD identity, based on sign-in name. For more information about assigning RBAC roles with Azure CLI, see [Manage access by using RBAC and Azure CLI](#).

Before you run the following sample script, remember to replace placeholder values, including brackets, with your own values.

```
#Assign the built-in role to the target identity: Storage File Data SMB Share Reader, Storage File Data SMB Share Contributor, Storage File Data SMB Share Elevated Contributor
az role assignment create --role "<role-name>" --assignee <user-principal-name> --scope
"/subscriptions/<subscription-id>/resourceGroups/<resource-group>/providers/Microsoft.Storage/storageAccounts/<storage-account>/fileServices/default/fileshares/<share-name>"
```

Configure NTFS permissions over SMB

After you assign share-level permissions with RBAC, you must assign proper NTFS permissions at the root, directory, or file level. Think of share-level permissions as the high-level gatekeeper that determines whether a user can access the share. Whereas NTFS permissions act at a more granular level to determine what operations the user can do at the directory or file level.

Azure Files supports the full set of NTFS basic and advanced permissions. You can view and configure NTFS permissions on directories and files in an Azure file share by mounting the share and then using Windows File Explorer or running the Windows [icacls](#) or [Set-ACL](#) command.

To configure NTFS with superuser permissions, you must mount the share by using your storage account key from your domain-joined VM. Follow the instructions in the next section to mount an Azure file share from the command prompt and to configure NTFS permissions accordingly.

The following sets of permissions are supported on the root directory of a file share:

- BUILTIN\Administrators:(OI)(CI)(F)
- NT AUTHORITY\SYSTEM:(OI)(CI)(F)
- BUILTIN\Users:(RX)
- BUILTIN\Users:(OI)(CI)(IO)(GR,GE)
- NT AUTHORITY\Authenticated Users:(OI)(CI)(M)
- NT AUTHORITY\SYSTEM:(F)
- CREATOR OWNER:(OI)(CI)(IO)(F)

Configure NTFS permissions with icacls

Use the following Windows command to grant full permissions to all directories and files under the file share, including the root directory. Remember to replace the placeholder values in the example with your own values.

```
icacls <mounted-drive-letter>: /grant <user-email>:(f)
```

For more information on how to use icacls to set NTFS permissions and on the different types of supported permissions, see [the command-line reference for icacls](#).

Mount a file share from the command prompt

Use the Windows **net use** command to mount the Azure file share. Remember to replace the placeholder values in the following example with your own values. For more information about mounting file shares, see [Use an Azure file share with Windows](#).

```
net use <desired-drive-letter>: \\<storage-account-name>.file.core.windows.net\<share-name> <storage-account-key> /user:Azure\<storage-account-name>
```

Configure NTFS permissions with Windows File Explorer

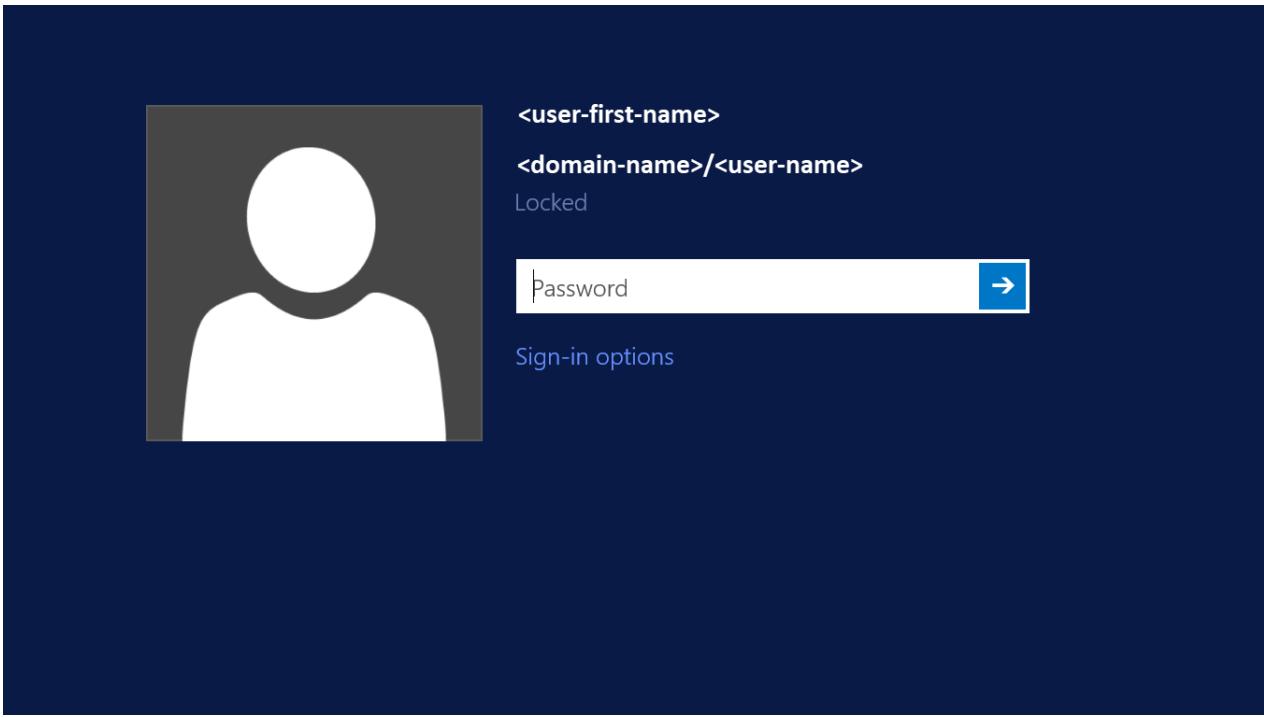
Use Windows File Explorer to grant full permission to all directories and files under the file share, including the root directory.

1. Open Windows File Explorer and right click on the file/directory and select **Properties**
2. Click on the **Security** tab
3. Click on **Edit...** button to change permissions
4. You can change the permission of existing users, or click on **Add...** to grant permissions to new users
5. In the prompt window for adding new users, enter the target user name you want to grant permission to in the **Enter the object names to select** box, and click on **Check Names** to find the full UPN name of the target user.
6. Click on **OK**
7. In the Security tab, select all permissions you want to grant to the newly add user
8. Click on **Apply**

Mount a file share from a domain-joined VM

The following process verifies that your file share and access permissions were set up correctly and that you can access an Azure File share from a domain-joined VM:

Sign in to the VM by using the Azure AD identity to which you have granted permissions, as shown in the following image. If you have enabled AD authentication for Azure Files, use the AD credential. For Azure AD DS authentication, log in with Azure AD credential.



Use the following command to mount the Azure file share. Remember to replace the placeholder values with your own values. Because you've been authenticated, you don't need to provide the storage account key, the AD credentials, or the Azure AD credentials. Single sign-on experience is supported for authentication with either AD or Azure AD DS.

```
net use <desired-drive-letter>: \\<storage-account-name>.file.core.windows.net\<share-name>
```

You have now successfully enabled Azure AD DS authentication over SMB and assigned a custom role that provides access to an Azure file share with an Azure AD identity. To grant additional users access to your file share, follow the instructions in the [Assign access permissions](#) to use an identity and [Configure NTFS permissions over SMB sections](#).

Next steps

For more information about Azure Files and how to use Azure AD over SMB, see these resources:

- [Overview of Azure Files identity-based authentication support for SMB access](#)
- [FAQ](#)

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 sessions:

- Session 1: 200 HTTP, Tunnel to config.edge.skygate.ms:443
- Session 2: 200 HTTP, Tunnel to client-office365-tas.msedge.net:443
- Session 3: 200 HTTP, Tunnel to blob.core.windows.net:443

 The details pane for Session 3 displays the ClientHello parameters. A red box highlights the 'Version: 3.3 (TLS 1.2)' entry. Below it, the 'Ciphers' section lists various cipher suites supported by the server.

Cipher Suite	Description
[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
[009F]	TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
[009E]	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
[009D]	TLS_RSA_WITH_AES_256_GCM_SHA384
[009C]	TLS_RSA_WITH_AES_128_GCM_SHA256
[003D]	TLS_RSA_WITH_AES_256_CBC_SHA256

See also

- [Transport Layer Security \(TLS\)](#)
- [PCI compliance on TLS](#)
- [Enable TLS in Java client](#)

Troubleshoot Azure Files problems in Windows

2/25/2020 • 13 minutes to read • [Edit Online](#)

This article lists common problems that are related to Microsoft Azure Files when you connect from Windows clients. It also provides possible causes and resolutions for these problems. In addition to the troubleshooting steps in this article, you can also use [AzFileDiagnostics](#) to ensure that the Windows client environment has correct prerequisites. AzFileDiagnostics automates detection of most of the symptoms mentioned in this article and helps set up your environment to get optimal performance. You can also find this information in the [Azure Files shares Troubleshooter](#) that provides steps to assist you with problems connecting/mapping/mounting Azure Files shares.

Error 5 when you mount an Azure file share

When you try to mount a file share, you might receive the following error:

- System error 5 has occurred. Access is denied.

Cause 1: Unencrypted communication channel

For security reasons, connections to Azure file shares are blocked if the communication channel isn't encrypted and if the connection attempt isn't made from the same datacenter where the Azure file shares reside.

Unencrypted connections within the same datacenter can also be blocked if the [Secure transfer required](#) setting is enabled on the storage account. An encrypted communication channel is provided only if the user's client OS supports SMB encryption.

Windows 8, Windows Server 2012, and later versions of each system negotiate requests that include SMB 3.0, which supports encryption.

Solution for cause 1

1. Connect from a client that supports SMB encryption (Windows 8, Windows Server 2012 or later) or connect from a virtual machine in the same datacenter as the Azure storage account that is used for the Azure file share.
2. Verify the [Secure transfer required](#) setting is disabled on the storage account if the client does not support SMB encryption.

Cause 2: Virtual network or firewall rules are enabled on the storage account

If virtual network (VNET) and firewall rules are configured on the storage account, network traffic will be denied access unless the client IP address or virtual network is allowed access.

Solution for cause 2

Verify virtual network and firewall rules are configured properly on the storage account. To test if virtual network or firewall rules is causing the issue, temporarily change the setting on the storage account to **Allow access from all networks**. To learn more, see [Configure Azure Storage firewalls and virtual networks](#).

Error 53, Error 67, or Error 87 when you mount or unmount an Azure file share

When you try to mount a file share from on-premises or from a different datacenter, you might receive the following errors:

- System error 53 has occurred. The network path was not found.
- System error 67 has occurred. The network name cannot be found.

- System error 87 has occurred. The parameter is incorrect.

Cause 1: Port 445 is blocked

System error 53 or system error 67 can occur if port 445 outbound communication to an Azure Files datacenter is blocked. To see the summary of ISPs that allow or disallow access from port 445, go to [TechNet](#).

To check if your firewall or ISP is blocking port 445, use the [AzFileDiagnostics](#) tool or [Test-NetConnection](#) cmdlet.

To use the [Test-NetConnection](#) cmdlet, the Azure PowerShell module must be installed, see [Install Azure PowerShell module](#) for more information. Remember to replace `<your-storage-account-name>` and `<your-resource-group-name>` with the relevant names for your storage account.

```
$resourceGroupName = "<your-resource-group-name>"  
$storageAccountName = "<your-storage-account-name>"  
  
# This command requires you to be logged into your Azure account, run Login-AzAccount if you haven't  
# already logged in.  
$storageAccount = Get-AzStorageAccount -ResourceGroupName $resourceGroupName -Name $storageAccountName  
  
# The ComputerName, or host, is <storage-account>.file.core.windows.net for Azure Public Regions.  
# $storageAccount.Context.FileEndpoint is used because non-Public Azure regions, such as sovereign clouds  
# or Azure Stack deployments, will have different hosts for Azure file shares (and other storage resources).  
Test-NetConnection -ComputerName ([System.Uri]::new($storageAccount.Context.FileEndPoint).Host) -Port 445
```

If the connection was successful, you should see the following output:

```
ComputerName      : <your-storage-account-name>  
RemoteAddress    : <storage-account-ip-address>  
RemotePort       : 445  
InterfaceAlias   : <your-network-interface>  
SourceAddress    : <your-ip-address>  
TcpTestSucceeded : True
```

NOTE

The above command returns the current IP address of the storage account. This IP address is not guaranteed to remain the same, and may change at any time. Do not hardcode this IP address into any scripts, or into a firewall configuration.

Solution for cause 1

Solution 1 - Use Azure File Sync

Azure File Sync can transform your on-premises Windows Server into a quick cache of your Azure file share. You can use any protocol that's available on Windows Server to access your data locally, including SMB, NFS, and FTPS. Azure File Sync works over port 443 and can thus be used as a workaround to access Azure Files from clients that have port 445 blocked. [Learn how to setup Azure File Sync](#).

Solution 2 - Use VPN

By Setting up a VPN to your specific Storage Account, the traffic will go through a secure tunnel as opposed to over the internet. Follow the [instructions to setup VPN](#) to access Azure Files from Windows.

Solution 3 - Unblock port 445 with help of your ISP/IT Admin

Work with your IT department or ISP to open port 445 outbound to [Azure IP ranges](#).

Solution 4 - Use REST API based tools like Storage Explorer/Powershell

Azure Files also supports REST in addition to SMB. REST access works over port 443 (standard tcp). There are various tools that are written using REST API which enable rich UI experience. [Storage Explorer](#) is one of them. [Download and Install Storage Explorer](#) and connect to your file share backed by Azure Files. You can also use

[PowerShell](#) which also uses REST API.

Cause 2: NTLMv1 is enabled

System error 53 or system error 87 can occur if NTLMv1 communication is enabled on the client. Azure Files supports only NTLMv2 authentication. Having NTLMv1 enabled creates a less-secure client. Therefore, communication is blocked for Azure Files.

To determine whether this is the cause of the error, verify that the following registry subkey is set to a value of 3:

HKLM\SYSTEM\CurrentControlSet\Control\Lsa > LmCompatibilityLevel

For more information, see the [LmCompatibilityLevel](#) topic on TechNet.

Solution for cause 2

Revert the **LmCompatibilityLevel** value to the default value of 3 in the following registry subkey:

HKLM\SYSTEM\CurrentControlSet\Control\Lsa

Error 1816 "Not enough quota is available to process this command" when you copy to an Azure file share

Cause

Error 1816 happens when you reach the upper limit of concurrent open handles that are allowed for a file on the computer where the file share is being mounted.

Solution

Reduce the number of concurrent open handles by closing some handles, and then retry. For more information, see [Microsoft Azure Storage performance and scalability checklist](#).

To view open handles for a file share, directory or file, use the [Get-AzStorageFileHandle](#) PowerShell cmdlet.

To close open handles for a file share, directory or file, use the [Close-AzStorageFileHandle](#) PowerShell cmdlet.

NOTE

The Get-AzStorageFileHandle and Close-AzStorageFileHandle cmdlets are included in Az PowerShell module version 2.4 or later. To install the latest Az PowerShell module, see [Install the Azure PowerShell module](#).

Error "No access" when you try to access or delete an Azure File Share

When you try to access or delete an Azure file share in the portal, you may receive the following error:

No access

Error code: 403

Cause 1: Virtual network or firewall rules are enabled on the storage account

Solution for cause 1

Verify virtual network and firewall rules are configured properly on the storage account. To test if virtual network or firewall rules is causing the issue, temporarily change the setting on the storage account to **Allow access from all networks**. To learn more, see [Configure Azure Storage firewalls and virtual networks](#).

Cause 2: Your user account does not have access to the storage account

Solution for cause 2

Browse to the storage account where the Azure file share is located, click **Access control (IAM)** and verify your user account has access to the storage account. To learn more, see [How to secure your storage account with Role-based access control](#).

Based Access Control (RBAC).

Unable to delete a file or directory in an Azure file share

When you try to delete a file, you may receive the following error:

The specified resource is marked for deletion by an SMB client.

Cause

This issue typically occurs if the file or directory has an open handle.

Solution

If the SMB clients have closed all open handles and the issue continues to occur, perform the following:

- Use the [Get-AzStorageFileHandle](#) PowerShell cmdlet to view open handles.
- Use the [Close-AzStorageFileHandle](#) PowerShell cmdlet to close open handles.

NOTE

The Get-AzStorageFileHandle and Close-AzStorageFileHandle cmdlets are included in Az PowerShell module version 2.4 or later. To install the latest Az PowerShell module, see [Install the Azure PowerShell module](#).

Slow file copying to and from Azure Files in Windows

You might see slow performance when you try to transfer files to the Azure File service.

- If you don't have a specific minimum I/O size requirement, we recommend that you use 1 MiB as the I/O size for optimal performance.
- If you know the final size of a file that you are extending with writes, and your software doesn't have compatibility problems when the unwritten tail on the file contains zeros, then set the file size in advance instead of making every write an extending write.
- Use the right copy method:
 - Use [AzCopy](#) for any transfer between two file shares.
 - Use [Robocopy](#) between file shares on an on-premises computer.

Considerations for Windows 8.1 or Windows Server 2012 R2

For clients that are running Windows 8.1 or Windows Server 2012 R2, make sure that the [KB3114025](#) hotfix is installed. This hotfix improves the performance of create and close handles.

You can run the following script to check whether the hotfix has been installed:

```
reg query HKLM\SYSTEM\CurrentControlSet\Services\LanmanWorkstation\Parameters\Policies
```

If hotfix is installed, the following output is displayed:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\LanmanWorkstation\Parameters\Policies {96c345ef-3cac-477b-8fcd-bea1a564241c} REG_DWORD 0x1
```

NOTE

Windows Server 2012 R2 images in Azure Marketplace have hotfix KB3114025 installed by default, starting in December 2015.

No folder with a drive letter in "My Computer" or "This PC"

If you map an Azure file share as an administrator by using net use, the share appears to be missing.

Cause

By default, Windows File Explorer does not run as an administrator. If you run net use from an administrative command prompt, you map the network drive as an administrator. Because mapped drives are user-centric, the user account that is logged in does not display the drives if they are mounted under a different user account.

Solution

Mount the share from a non-administrator command line. Alternatively, you can follow [this TechNet topic](#) to configure the **EnableLinkedConnections** registry value.

Net use command fails if the storage account contains a forward slash

Cause

The net use command interprets a forward slash (/) as a command-line option. If your user account name starts with a forward slash, the drive mapping fails.

Solution

You can use either of the following steps to work around the problem:

- Run the following PowerShell command:

```
New-SmbMapping -LocalPath y: -RemotePath \\server\share -UserName accountName -Password "password can contain / and \ etc"
```

From a batch file, you can run the command this way:

```
Echo new-smbMapping ... | powershell -command -
```

- Put double quotation marks around the key to work around this problem--unless the forward slash is the first character. If it is, either use the interactive mode and enter your password separately or regenerate your keys to get a key that doesn't start with a forward slash.

Application or service cannot access a mounted Azure Files drive

Cause

Drives are mounted per user. If your application or service is running under a different user account than the one that mounted the drive, the application will not see the drive.

Solution

Use one of the following solutions:

- Mount the drive from the same user account that contains the application. You can use a tool such as PsExec.
- Pass the storage account name and key in the user name and password parameters of the net use command.
- Use the cmdkey command to add the credentials into Credential Manager. Perform this from a command line under the service account context, either through an interactive login or by using `runas`.

```
cmdkey /add:<storage-account-name>.file.core.windows.net /user:AZURE\<storage-account-name> /pass:<storage-account-key>
```

- Map the share directly without using a mapped drive letter. Some applications may not reconnect to the drive letter properly, so using the full UNC path may be more reliable.

```
net use * \\storage-account-name.file.core.windows.net\share
```

After you follow these instructions, you might receive the following error message when you run net use for the system/network service account: "System error 1312 has occurred. A specified logon session does not exist. It may already have been terminated." If this occurs, make sure that the username that is passed to net use includes domain information (for example: "[storage account name].file.core.windows.net").

Error "You are copying a file to a destination that does not support encryption"

When a file is copied over the network, the file is decrypted on the source computer, transmitted in plaintext, and re-encrypted at the destination. However, you might see the following error when you're trying to copy an encrypted file: "You are copying the file to a destination that does not support encryption."

Cause

This problem can occur if you are using Encrypting File System (EFS). BitLocker-encrypted files can be copied to Azure Files. However, Azure Files does not support NTFS EFS.

Workaround

To copy a file over the network, you must first decrypt it. Use one of the following methods:

- Use the **copy /d** command. It allows the encrypted files to be saved as decrypted files at the destination.
- Set the following registry key:
 - Path = HKLM\Software\Policies\Microsoft\Windows\System
 - Value type = DWORD
 - Name = CopyFileAllowDecryptedRemoteDestination
 - Value = 1

Be aware that setting the registry key affects all copy operations that are made to network shares.

Slow enumeration of files and folders

Cause

This problem can occur if there is no enough cache on client machine for large directories.

Solution

To resolve this problem, adjusting the **DirectoryCacheEntrySizeMax** registry value to allow caching of larger directory listings in the client machine:

- Location: HKLM\System\CCS\Services\Lanmanworkstation\Parameters
- Value mane: DirectoryCacheEntrySizeMax
- Value type:DWORD

For example, you can set it to 0x100000 and see if the performance become better.

Error AadDsTenantNotFound in enabling Azure Active Directory Domain Service (AAD DS) authentication for Azure Files "Unable to locate active tenants with tenant Id aad-tenant-id"

Cause

Error AadDsTenantNotFound happens when you try to [enable Azure Active Directory Domain Services \(Azure AD DS\) authentication on Azure Files](#) on a storage account where [AAD Domain Service\(AAD DS\)](#) is not created on the AAD tenant of the associated subscription.

Solution

Enable AAD DS on the AAD tenant of the subscription that your storage account is deployed to. You need administrator privileges of the AAD tenant to create a managed domain. If you aren't the administrator of the Azure AD tenant, contact the administrator and follow the step-by-step guidance to [Enable Azure Active Directory Domain Services using the Azure portal](#).

Error ConditionHeadersNotSupported from a Web Application using Azure Files from Browser

The ConditionHeadersNotSupported error occurs when accessing content hosted in Azure Files through an application that makes use of conditional headers, such as a web browser, access fails. The error states that condition headers are not supported.



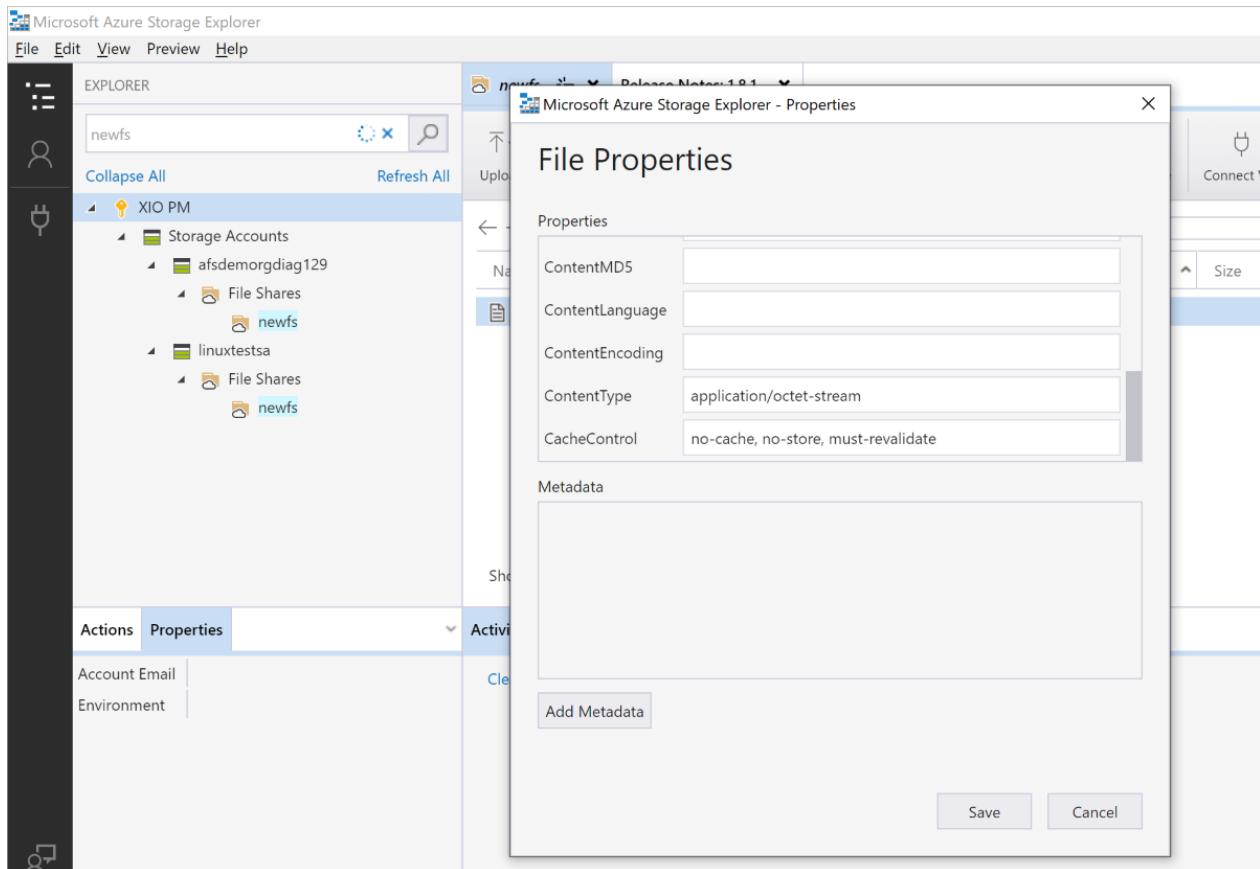
```
<?xml version="1.0" encoding="ISO-8859-1"?>
- <Error>
  <Code>ConditionHeadersNotSupported</Code>
  <Message>Condition headers are not supported. RequestId:d24c7b15-601a-0050-7427-0f5a96000000 Time:2019-05-20T16:20:47.6322722Z</Message>
</Error>
```

Cause

Conditional headers are not yet supported. Applications implementing them will need to request the full file every time the file is accessed.

Workaround

When a new file is uploaded, the cache-control property by default is "no-cache". To force the application to request the file every time, the file's cache-control property needs to be updated from "no-cache" to "no-cache, no-store, must-revalidate". This can be achieved using [Azure Storage Explorer](#).



Error 'System error 1359 has occurred. An internal error' received over SMB access to file shares with Azure Active Directory Domain Service

(AAD DS) authentication enabled

Cause

Error 'System error 1359 has occurred. An internal error' happens when you try to connect to your file share with AAD DS authentication enabled against an AAD DS with domain DNS name starting with a numeric character. For example, if your AAD DS Domain DNS name is "1domain", you will get this error when attempting to mount the file share using AAD credentials.

Solution

Currently, you can consider redeploying your AAD DS using a new domain DNS name that applies with the rules below:

- Names cannot begin with a numeric character.
- Names must be from 3 to 63 characters long.

Need help? Contact support.

If you still need help, [contact support](#) to get your problem resolved quickly.

Troubleshoot Azure Files problems in Linux

1/22/2020 • 12 minutes to read • [Edit Online](#)

This article lists common problems that are related to Azure Files when you connect from Linux clients. It also provides possible causes and resolutions for these problems.

In addition to the troubleshooting steps in this article, you can use [AzFileDiagnostics](#) to ensure that the Linux client has correct prerequisites. AzFileDiagnostics automates the detection of most of the symptoms mentioned in this article. It helps set up your environment to get optimal performance. You can also find this information in the [Azure Files shares troubleshooter](#). The troubleshooter provides steps to help you with problems connecting, mapping, and mounting Azure Files shares.

Cannot connect to or mount an Azure file share

Cause

Common causes for this problem are:

- You're using an incompatible Linux distribution client. We recommend that you use the following Linux distributions to connect to an Azure file share:

	SMB 2.1 (MOUNTS ON VMS WITHIN THE SAME AZURE REGION)	SMB 3.0 (MOUNTS FROM ON-PREMISES AND CROSS-REGION)
Ubuntu Server	14.04+	16.04+
RHEL	7+	7.5+
CentOS	7+	7.5+
Debian	8+	
openSUSE	13.2+	42.3+
SUSE Linux Enterprise Server	12	12 SP3+

- CIFS utilities (`cifs-utils`) are not installed on the client.
- The minimum SMB/CIFS version, 2.1, is not installed on the client.
- SMB 3.0 encryption is not supported on the client. The preceding table provides a list of Linux distributions that support mounting from on-premises and cross-region using encryption. Other distributions require kernel 4.11 and later versions.
- You're trying to connect to a storage account over TCP port 445, which is not supported.
- You're trying to connect to an Azure file share from an Azure VM, and the VM is not in the same region as the storage account.
- If the [Secure transfer required](#) setting is enabled on the storage account, Azure Files will allow only connections that use SMB 3.0 with encryption.

Solution

To resolve the problem, use the [troubleshooting tool for Azure Files mounting errors on Linux](#). This tool:

- Helps you to validate the client running environment.
- Detects the incompatible client configuration that would cause access failure for Azure Files.
- Gives prescriptive guidance on self-fixing.
- Collects the diagnostics traces.

"Mount error(13): Permission denied" when you mount an Azure file share

Cause 1: Unencrypted communication channel

For security reasons, connections to Azure file shares are blocked if the communication channel isn't encrypted and if the connection attempt isn't made from the same datacenter where the Azure file shares reside.

Unencrypted connections within the same datacenter can also be blocked if the [Secure transfer required](#) setting is enabled on the storage account. An encrypted communication channel is provided only if the user's client OS supports SMB encryption.

To learn more, see [Prerequisites for mounting an Azure file share with Linux and the cifs-utils package](#).

Solution for cause 1

1. Connect from a client that supports SMB encryption or connect from a virtual machine in the same datacenter as the Azure storage account that is used for the Azure file share.
2. Verify the [Secure transfer required](#) setting is disabled on the storage account if the client does not support SMB encryption.

Cause 2: Virtual network or firewall rules are enabled on the storage account

If virtual network (VNET) and firewall rules are configured on the storage account, network traffic will be denied access unless the client IP address or virtual network is allowed access.

Solution for cause 2

Verify virtual network and firewall rules are configured properly on the storage account. To test if virtual network or firewall rules is causing the issue, temporarily change the setting on the storage account to **Allow access from all networks**. To learn more, see [Configure Azure Storage firewalls and virtual networks](#).

"[permission denied] Disk quota exceeded" when you try to open a file

In Linux, you receive an error message that resembles the following:

<filename> [permission denied] Disk quota exceeded

Cause

You have reached the upper limit of concurrent open handles that are allowed for a file.

There is a quota of 2,000 open handles on a single file. When you have 2,000 open handles, an error message is displayed that says the quota is reached.

Solution

Reduce the number of concurrent open handles by closing some handles, and then retry the operation.

To view open handles for a file share, directory or file, use the [Get-AzStorageFileHandle](#) PowerShell cmdlet.

To close open handles for a file share, directory or file, use the [Close-AzStorageFileHandle](#) PowerShell cmdlet.

NOTE

The Get-AzStorageFileHandle and Close-AzStorageFileHandle cmdlets are included in Az PowerShell module version 2.4 or later. To install the latest Az PowerShell module, see [Install the Azure PowerShell module](#).

Slow file copying to and from Azure Files in Linux

- If you don't have a specific minimum I/O size requirement, we recommend that you use 1 MiB as the I/O size for optimal performance.
- Use the right copy method:
 - Use [AzCopy](#) for any transfer between two file shares.
 - Using cp or dd with parallel could improve copy speed, the number of threads depends on your use case and workload. The following examples use six:
 - cp example (cp will use the default block size of the file system as the chunk size):

```
find * -type f | parallel --will-cite -j 6 cp {} /mnt/premium/ & .
```
 - dd example (this command explicitly sets chunk size to 1 MiB):

```
find * -type f | parallel --will-cite-j 6 dd if={} of=/mnt/share/{}/ bs=1M
```
 - Open source third party tools such as:
 - [GNU Parallel](#).
 - [Fpart](#) - Sorts files and packs them into partitions.
 - [Fpsync](#) - Uses Fpart and a copy tool to spawn multiple instances to migrate data from src_dir to dst_url.
 - [Multi](#) - Multi-threaded cp and md5sum based on GNU coreutils.
- Setting the file size in advance, instead of making every write an extending write, helps improve copy speed in scenarios where the file size is known. If extending writes need to be avoided, you can set a destination file size with `truncate - size <size><file>` command. After that, `dd if=<source> of=<target> bs=1M conv=notrunc` command will copy a source file without having to repeatedly update the size of the target file. For example, you can set the destination file size for every file you want to copy (assume a share is mounted under /mnt/share):
 - ```
$ for i in ` find * -type f`; do truncate --size `stat -c%$i` /mnt/share/$i; done
```
  - and then - copy files without extending writes in parallel:

```
$find * -type f | parallel -j6 dd if={} of =/mnt/share/{}/ bs=1M conv=notrunc
```

## "Mount error(115): Operation now in progress" when you mount Azure Files by using SMB 3.0

#### **Cause**

Some Linux distributions don't yet support encryption features in SMB 3.0. Users might receive a "115" error message if they try to mount Azure Files by using SMB 3.0 because of a missing feature. SMB 3.0 with full encryption is supported only when you're using Ubuntu 16.04 or later.

#### **Solution**

The encryption feature for SMB 3.0 for Linux was introduced in the 4.11 kernel. This feature enables mounting of an Azure file share from on-premises or from a different Azure region. Some Linux distributions may have backported changes from the 4.11 kernel to older versions of the Linux kernel which they maintain. To assist in determining if your version of Linux supports SMB 3.0 with encryption, consult with [Use Azure Files with Linux](#).

If your Linux SMB client doesn't support encryption, mount Azure Files by using SMB 2.1 from an Azure Linux VM that's in the same datacenter as the file share. Verify that the [Secure transfer required](#) setting is disabled on the storage account.

# Error "No access" when you try to access or delete an Azure File Share

When you try to access or delete an Azure file share in the portal, you may receive the following error:

No access

Error code: 403

## Cause 1: Virtual network or firewall rules are enabled on the storage account

### Solution for cause 1

Verify virtual network and firewall rules are configured properly on the storage account. To test if virtual network or firewall rules is causing the issue, temporarily change the setting on the storage account to **Allow access from all networks**. To learn more, see [Configure Azure Storage firewalls and virtual networks](#).

## Cause 2: Your user account does not have access to the storage account

### Solution for cause 2

Browse to the storage account where the Azure file share is located, click **Access control (IAM)** and verify your user account has access to the storage account. To learn more, see [How to secure your storage account with Role-Based Access Control \(RBAC\)](#).

# Unable to delete a file or directory in an Azure file share

## Cause

This issue typically occurs if the file or directory has an open handle.

## Solution

If the SMB clients have closed all open handles and the issue continues to occur, perform the following:

- Use the [Get-AzStorageFileHandle](#) PowerShell cmdlet to view open handles.
- Use the [Close-AzStorageFileHandle](#) PowerShell cmdlet to close open handles.

### NOTE

The Get-AzStorageFileHandle and Close-AzStorageFileHandle cmdlets are included in Az PowerShell module version 2.4 or later. To install the latest Az PowerShell module, see [Install the Azure PowerShell module](#).

# Slow performance on an Azure file share mounted on a Linux VM

## Cause 1: Caching

One possible cause of slow performance is disabled caching. Caching can be useful if you are accessing a file repeatedly, otherwise, it can be an overhead. Check if you are using the cache before disabling it.

### Solution for cause 1

To check whether caching is disabled, look for the **cache=** entry.

**Cache=None** indicates that caching is disabled. Remount the share by using the default mount command or by explicitly adding the **cache=strict** option to the mount command to ensure that default caching or "strict" caching mode is enabled.

In some scenarios, the **serverino** mount option can cause the **ls** command to run stat against every directory entry. This behavior results in performance degradation when you're listing a large directory. You can check the mount options in your **/etc/fstab** entry:

```
//azureuser.file.core.windows.net/cifs /cifs cifs
vers=2.1,serverino,username=xxx,password=xxx,dir_mode=0777,file_mode=0777
```

You can also check whether the correct options are being used by running the **sudo mount | grep cifs** command and checking its output. The following is example output:

```
//azureuser.file.core.windows.net/cifs on /cifs type cifs
(rw,relatime,vers=2.1,sec=ntlmssp,cache=strict,username=xxx,domain=X,uid=0,noforceuid,gid=0,noforcegid,addr=1
92.168.10.1,file_mode=0777,
dir_mode=0777,persistenthandles,nounix,serverino,mapposix,rsize=1048576,wsize=1048576,actimeo=1)
```

If the **cache=strict** or **serverino** option is not present, unmount and mount Azure Files again by running the mount command from the [documentation](#). Then, recheck that the **/etc/fstab** entry has the correct options.

### Cause 2: Throttling

It is possible you are experiencing throttling and your requests are being sent to a queue. You can verify this by leveraging [Azure Storage metrics in Azure Monitor](#).

### Solution for cause 2

Ensure your app is within the [Azure Files scale targets](#).

## Time stamps were lost in copying files from Windows to Linux

On Linux/Unix platforms, the **cp -p** command fails if different users own file 1 and file 2.

### Cause

The force flag **f** in COPYFILE results in executing **cp -p -f** on Unix. This command also fails to preserve the time stamp of the file that you don't own.

### Workaround

Use the storage account user for copying the files:

- Useadd : [storage account name]
- Passwd [storage account name]
- Su [storage account name]
- Cp -p filename.txt /share

## ls: cannot access '<path>': Input/output error

When you try to list files in an Azure file share by using the ls command, the command hangs when listing files. You get the following error:

### ls: cannot access '<path>': Input/output error

### Solution

Upgrade the Linux kernel to the following versions that have a fix for this problem:

- 4.4.87+
- 4.9.48+
- 4.12.11+
- All versions that are greater than or equal to 4.13

## Cannot create symbolic links - ln: failed to create symbolic link 't': Operation not supported

### Cause

By default, mounting Azure file shares on Linux by using CIFS doesn't enable support for symbolic links

(symlinks). You see an error like this:

```
ln -s linked -n t
ln: failed to create symbolic link 't': Operation not supported
```

## Solution

The Linux CIFS client doesn't support creation of Windows-style symbolic links over the SMB 2 or 3 protocol. Currently, the Linux client supports another style of symbolic links called [Minshall+French symlinks](#) for both create and follow operations. Customers who need symbolic links can use the "mfsymlinks" mount option. We recommend "mfsymlinks" because it's also the format that Macs use.

To use symlinks, add the following to the end of your CIFS mount command:

```
,mfsymlinks
```

So the command looks something like:

```
sudo mount -t cifs //<storage-account-name>.file.core.windows.net/<share-name> <mount-point> -o vers=<smb-version>,username=<storage-account-name>,password=<storage-account-key>,dir_mode=0777,file_mode=0777,serverino,mfsymlinks
```

You can then create symlinks as suggested on the [wiki](#).

## Error ConditionHeadersNotSupported from a Web Application using Azure Files from Browser

The ConditionHeadersNotSupported error occurs when accessing content hosted in Azure Files through an application that makes use of conditional headers, such as a web browser, access fails. The error states that condition headers are not supported.

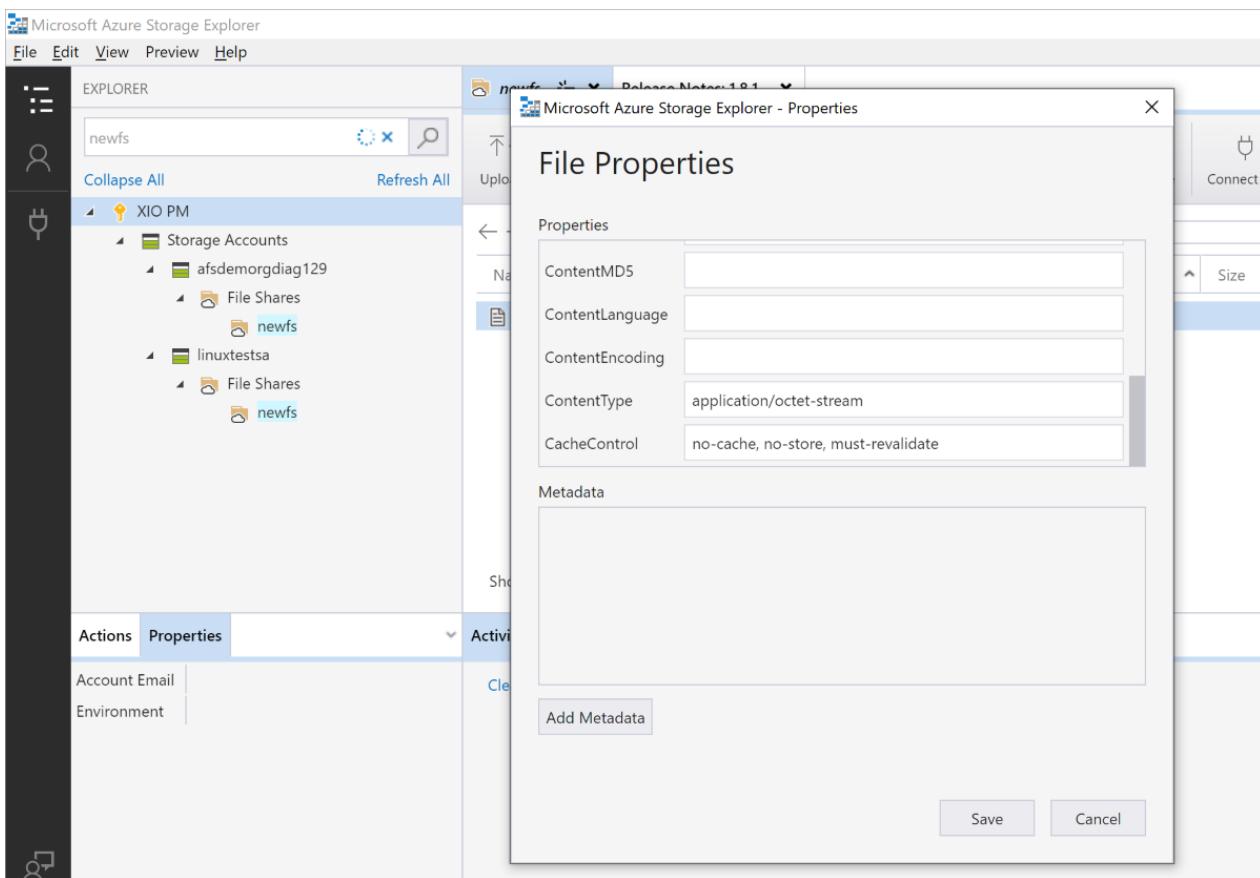


## Cause

Conditional headers are not yet supported. Applications implementing them will need to request the full file every time the file is accessed.

## Workaround

When a new file is uploaded, the cache-control property by default is "no-cache". To force the application to request the file every time, the file's cache-control property needs to be updated from "no-cache" to "no-cache, no-store, must-revalidate". This can be achieved using [Azure Storage Explorer](#).



## "Mount error(112): Host is down" because of a reconnection time-out

A "112" mount error occurs on the Linux client when the client has been idle for a long time. After an extended idle time, the client disconnects and the connection times out.

### Cause

The connection can be idle for the following reasons:

- Network communication failures that prevent re-establishing a TCP connection to the server when the default "soft" mount option is used
- Recent reconnection fixes that are not present in older kernels

### Solution

This reconnection problem in the Linux kernel is now fixed as part of the following changes:

- [Fix reconnect to not defer smb3 session reconnect long after socket reconnect](#)
- [Call echo service immediately after socket reconnect](#)
- [CIFS: Fix a possible memory corruption during reconnect](#)
- [CIFS: Fix a possible double locking of mutex during reconnect \(for kernel v4.9 and later\)](#)

However, these changes might not be ported yet to all the Linux distributions. If you're using a popular Linux distribution, you can check on the [Use Azure Files with Linux](#) to see which version of your distribution has the necessary kernel changes.

### Workaround

You can work around this problem by specifying a hard mount. A hard mount forces the client to wait until a connection is established or until it's explicitly interrupted. You can use it to prevent errors because of network time-outs. However, this workaround might cause indefinite waits. Be prepared to stop connections as necessary.

If you can't upgrade to the latest kernel versions, you can work around this problem by keeping a file in the Azure file share that you write to every 30 seconds or less. This must be a write operation, such as rewriting the created

or modified date on the file. Otherwise, you might get cached results, and your operation might not trigger the reconnection.

## Need help? Contact support.

If you still need help, [contact support](#) to get your problem resolved quickly.

# Troubleshoot Azure Files performance issues

2/25/2020 • 6 minutes to read • [Edit Online](#)

This article lists some common problems related to Azure file shares. It provides potential causes and workarounds when these problems are encountered.

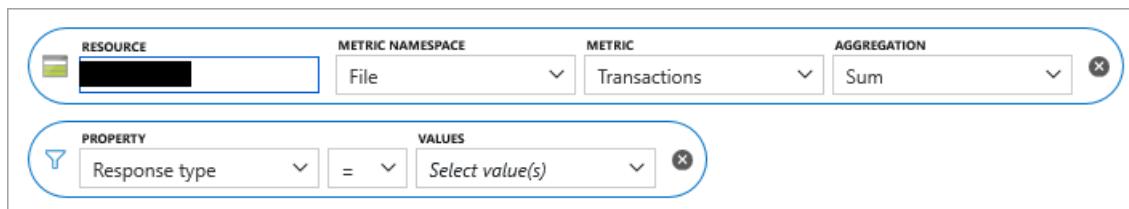
## High latency, low throughput, and general performance issues

### Cause 1: Share experiencing throttling

The default quota on a premium share is 100 GiB, which provides 100 baseline IOPS (with a potential to burst up to 300 for an hour). For more information about provisioning and its relationship to IOPS, see the [Provisioned shares](#) section of the planning guide.

To confirm if your share is being throttled, you can leverage Azure Metrics in the portal.

1. Sign in to the [Azure portal](#).
2. Select **All services** and then search for **Metrics**.
3. Select **Metrics**.
4. Select your storage account as the resource.
5. Select **File** as the metric namespace.
6. Select **Transactions** as the metric.
7. Add a filter for **ResponseType** and check to see if any requests have a response code of **SuccessWithThrottling** (for SMB) or **ClientThrottlingError** (for REST).



#### NOTE

To receive an alert if a file share is throttled, see [How to create an alert if a file share is throttled](#).

### Solution

- Increase share provisioned capacity by specifying a higher quota on your share.

### Cause 2: Metadata/namespace heavy workload

If the majority of your requests are metadata centric, (such as `createfile/openfile/closefile/queryinfo/querydirectory`) then the latency will be worse when compared to read/write operations.

To confirm if most of your requests are metadata centric, you can use the same steps as above. Except instead of adding a filter for **ResponseType**, add a filter for **API Name**.

The screenshot shows the Azure Metrics Explorer interface. At the top, there are four dropdown menus: 'RESOURCE' (with a redacted icon), 'METRIC NAMESPACE' (set to 'File'), 'METRIC' (set to 'Transactions'), and 'AGGREGATION' (set to 'Sum'). Below these are two more dropdowns: 'PROPERTY' (set to 'API name') and 'VALUES' (set to '= Select value(s)'). A small 'X' button is located to the right of the 'VALUES' dropdown.

## Workaround

- Check if the application can be modified to reduce the number of metadata operations.
- Add a VHD on the file share and mount VHD over SMB from the client to perform files operations against the data. This approach works for single writer and multiple readers scenarios and allows metadata operations to be local, offering performance similar to a local direct-attached storage.

## Cause 3: Single-threaded application

If the application being used by the customer is single-threaded, this can result in significantly lower IOPS/throughput than the maximum possible based on your provisioned share size.

## Solution

- Increase application parallelism by increasing the number of threads.
- Switch to applications where parallelism is possible. For example, for copy operations, customers could use AzCopy or RoboCopy from Windows clients or the **parallel** command on Linux clients.

## Very high latency for requests

### Cause

The client VM could be located in a different region than the file share.

### Solution

- Run the application from a VM that is located in the same region as the file share.

## Client unable to achieve maximum throughput supported by the network

One potential cause of this is a lack of SMB multi-channel support. Currently, Azure file shares only support single channel, so there is only one connection from the client VM to the server. This single connection is pegged to a single core on the client VM, so the maximum throughput achievable from a VM is bound by a single core.

## Workaround

- Obtaining a VM with a bigger core may help improve throughput.
- Running the client application from multiple VMs will increase throughput.
- Use REST APIs where possible.

## Throughput on Linux clients is significantly lower when compared to Windows clients.

### Cause

This is a known issue with the implementation of SMB client on Linux.

## Workaround

- Spread the load across multiple VMs.
- On the same VM, use multiple mount points with **nosharesock** option, and spread the load across these mount points.

- On Linux, try mounting with **nostrictsync** option to avoid forcing SMB flush on every **fsync** call. For Azure Files, this option does not interfere with data consistency, but may result in stale file metadata on directory listing (**ls -l** command). Directly querying metadata of file (**stat** command) will return the most up-to date file metadata.

## High latencies for metadata heavy workloads involving extensive open/close operations.

### Cause

Lack of support for directory leases.

### Workaround

- If possible, avoid excessive opening/closing handle on the same directory within a short period of time.
- For Linux VMs, increase the directory entry cache timeout by specifying **actimeo=<sec>** as a mount option. By default, it is one second, so a larger value like three or five might help.
- For Linux VMs, upgrade the kernel to 4.20 or higher.

## Low IOPS on CentOS/RHEL

### Cause

IO depth greater than one is not supported on CentOS/RHEL.

### Workaround

- Upgrade to CentOS 8 / RHEL 8.
- Change to Ubuntu.

## Slow file copying to and from Azure Files in Linux

If you are experiencing slow file copying to and from Azure Files, take a look at the [Slow file copying to and from Azure Files in Linux](#) section in the Linux troubleshooting guide.

## Jittery/saw-tooth pattern for IOPS

### Cause

Client application consistently exceeds baseline IOPS. Currently, there is no service side smoothing of the request load, so if the client exceeds baseline IOPS, it will get throttled by the service. That throttling can result in the client experiencing a jittery/saw-tooth IOPS pattern. In this case, average IOPS achieved by the client might be lower than the baseline IOPS.

### Workaround

- Reduce the request load from the client application, so that the share does not get throttled.
- Increase the quota of the share so that the share does not get throttled.

## Excessive DirectoryOpen/DirectoryClose calls

### Cause

If the number of DirectoryOpen/DirectoryClose calls is among the top API calls and you don't expect the client to be making that many calls, it may be an issue with the antivirus installed on the Azure client VM.

### Workaround

- A fix for this issue is available in the [April Platform Update for Windows](#).

# File creation is slower than expected

## Cause

Workloads that rely on creating a large number of files will not see a substantial difference between the performance of premium file shares and standard file shares.

## Workaround

- None.

# Slow performance from Windows 8.1 or Server 2012 R2

## Cause

Higher than expected latency accessing Azure Files for IO intensive workloads.

## Workaround

- Install the available [hotfix](#).

# How to create an alert if a file share is throttled

1. In the [Azure portal](#), click on **Monitor**.
2. Click **Alerts** and then click **+ New alert rule**.
3. Click **Select** to select the **storage account/file** resource that contains the file share that you want to alert on and then click **Done**. For example, if the storage account name is contoso, select the contoso/file resource.
4. Click **Add** to add a condition.
5. You will see a list of signals supported for the storage account, select the **Transactions** metric.
6. On the **Configure signal logic** blade, go to the **Response type** dimension, click the **Dimension values** drop-down and select **SuccessWithThrottling** (for SMB) or **ClientThrottlingError** (for REST).

### NOTE

If the SuccessWithThrottling or ClientThrottlingError dimension value is not listed, this means the resource has not been throttled. To add the dimension value, click the **+** beside the **Dimension values** drop-down, type **SuccessWithThrottling** or **ClientThrottlingError**, click **OK** and then repeat step #6.

7. Go to the **File Share** dimension, click the **Dimension values** drop-down and select the file share(s) that you want to alert on.

### NOTE

If the file share is a standard file share, the dimension values drop-down will be blank because per-share metrics are not available for standard file shares. Throttling alerts for standard file shares will be triggered if any file share within the storage account is throttled and the alert will not identify which file share was throttled. Since per-share metrics are not available for standard file shares, the recommendation is to have one file share per storage account.

8. Define the **alert parameters** (threshold, operator, aggregation granularity and frequency) which are used to evaluate the metric alert rule and click **Done**.

**TIP**

If you are using a static threshold, the metric chart can help determine a reasonable threshold if the file share is currently being throttled. If you are using a dynamic threshold, the metric chart will display the calculated thresholds based on recent data.

9. Add an **action group** (email, SMS, etc.) to the alert either by selecting an existing action group or creating a new action group.
10. Fill in the **Alert details** like **Alert rule name**, **Description** and **Severity**.
11. Click **Create alert rule** to create the alert.

To learn more about configuring alerts in Azure Monitor, see [Overview of alerts in Microsoft Azure](#).

# Troubleshoot Azure File Sync

2/5/2020 • 61 minutes to read • [Edit Online](#)

Use Azure File Sync to centralize your organization's file shares in Azure Files, while keeping the flexibility, performance, and compatibility of an on-premises file server. Azure File Sync transforms Windows Server into a quick cache of your Azure file share. You can use any protocol that's available on Windows Server to access your data locally, including SMB, NFS, and FTPS. You can have as many caches as you need across the world.

This article is designed to help you troubleshoot and resolve issues that you might encounter with your Azure File Sync deployment. We also describe how to collect important logs from the system if a deeper investigation of the issue is required. If you don't see the answer to your question, you can contact us through the following channels (in escalating order):

1. [Azure Storage Forum](#).
2. [Azure Files UserVoice](#).
3. Microsoft Support. To create a new support request, in the Azure portal, on the **Help** tab, select the **Help + support** button, and then select **New support request**.

I'm having an issue with Azure File Sync on my server (sync, cloud tiering, etc.). Should I remove and recreate my server endpoint?

No: removing a server endpoint isn't like rebooting a server! Removing and recreating the server endpoint is almost never an appropriate solution to fixing issues with sync, cloud tiering, or other aspects of Azure File Sync. Removing a server endpoint is a destructive operation. It may result in data loss in the case that tiered files exist outside of the server endpoint namespace. See [Why do tiered files exist outside of the server endpoint namespace](#) for more information. Or it may result in inaccessible files for tiered files that exist within the server endpoint namespace. These issues won't resolve when the server endpoint is recreated. Tiered files may exist within your server endpoint namespace even if you never had cloud tiering enabled. That's why recommend that you don't remove the server endpoint unless you would like to stop using Azure File Sync with this particular folder or have been explicitly instructed to do so by a Microsoft engineer. For more information on remove server endpoints, see [Remove a server endpoint](#).

## Agent installation and server registration

### Troubleshoot agent installation failures

If the Azure File Sync agent installation fails, at an elevated command prompt, run the following command to turn on logging during agent installation:

```
StorageSyncAgent.msi /l*v AFSInstaller.log
```

Review installer.log to determine the cause of the installation failure.

### Agent installation fails on Active Directory Domain Controller

If you try to install the sync agent on an Active Directory domain controller where the PDC role owner is on a Windows Server 2008 R2 or below OS version, you may hit the issue where the sync agent will fail to install.

To resolve, transfer the PDC role to another domain controller running Windows Server 2012 R2 or more recent, then install sync.

### Accessing a volume on Windows Server 2012 R2 fails with error: The parameter is incorrect

After creating a server endpoint on Windows Server 2012 R2, the following error occurs when accessing the volume:

driveletter:\ is not accessible.

The parameter is incorrect.

To resolve, install the latest updates for Windows Server 2012 R2 and restart the server.

### **Server Registration does not list all Azure Subscriptions**

When registering a server using ServerRegistration.exe, subscriptions are missing when you click the Azure Subscription drop-down.

This issue occurs because ServerRegistration.exe does not currently support multi-tenant environments. This issue will be fixed in a future Azure File Sync agent update.

To workaround this issue, use the following PowerShell commands to register the server:

```
Import-Module "C:\Program Files\Azure\StorageSyncAgent\StorageSync.Management.PowerShell.Cmdlets.dll"
Login-AzureRmStorageSync -SubscriptionID "<guid>" -TenantID "<guid>"
Register-AzureRmStorageSyncServer -SubscriptionId "<guid>" -ResourceGroupName "<string>" -
StorageSyncServiceName "<string>"
```

### **Server Registration displays the following message: "Pre-requisites are missing"**

This message appears if Az or AzureRM PowerShell module is not installed on PowerShell 5.1.

#### **NOTE**

ServerRegistration.exe does not support PowerShell 6.x. You can use the Register-AzStorageSyncServer cmdlet on PowerShell 6.x to register the server.

To install the Az or AzureRM module on PowerShell 5.1, perform the following steps:

1. Type **powershell** from an elevated command prompt and hit enter.
2. Install the latest Az or AzureRM module by following the documentation:
  - [Az module \(requires .NET 4.7.2\)](#)
  - [AzureRM module](#)
3. Run ServerRegistration.exe, and complete the wizard to register the server with a Storage Sync Service.

### **Server Registration displays the following message: "This server is already registered"**



This server is already registered

A server can be registered with only one Storage Sync Service at a time.

OK

This message appears if the server was previously registered with a Storage Sync Service. To unregister the server from the current Storage Sync Service and then register with a new Storage Sync Service, complete the steps that are described in [Unregister a server with Azure File Sync](#).

If the server is not listed under **Registered servers** in the Storage Sync Service, on the server that you want to unregister, run the following PowerShell commands:

```
Import-Module "C:\Program Files\Azure\StorageSyncAgent\StorageSync.Management.ServerCmdlets.dll"
Reset-StorageSyncServer
```

#### NOTE

If the server is part of a cluster, you can use the optional `Reset-StorageSyncServer -CleanClusterRegistration` parameter to also remove the cluster registration.

#### When I register a server, I see numerous "web site not trusted" responses. Why?

This issue occurs when the **Enhanced Internet Explorer Security** policy is enabled during server registration.

For more information about how to correctly disable the **Enhanced Internet Explorer Security** policy, see [Prepare Windows Server to use with Azure File Sync](#) and [How to deploy Azure File Sync](#).

#### Server is not listed under registered servers in the Azure portal

If a server is not listed under **Registered servers** for a Storage Sync Service:

1. Sign in to the server that you want to register.
2. Open File Explorer, and then go to the Storage Sync Agent installation directory (the default location is `C:\Program Files\Azure\StorageSyncAgent`).
3. Run `ServerRegistration.exe`, and complete the wizard to register the server with a Storage Sync Service.

## Sync group management

#### Cloud endpoint creation fails, with this error: "The specified Azure FileShare is already in use by a different CloudEndpoint"

This error occurs if the Azure file share is already in use by another cloud endpoint.

If you see this message and the Azure file share currently is not in use by a cloud endpoint, complete the following steps to clear the Azure File Sync metadata on the Azure file share:

**WARNING**

Deleting the metadata on an Azure file share that is currently in use by a cloud endpoint causes Azure File Sync operations to fail.

1. In the Azure portal, go to your Azure file share.
2. Right-click the Azure file share, and then select **Edit metadata**.
3. Right-click **SyncService**, and then select **Delete**.

**Cloud endpoint creation fails, with this error: "AuthorizationFailed"**

This error occurs if your user account doesn't have sufficient rights to create a cloud endpoint.

To create a cloud endpoint, your user account must have the following Microsoft Authorization permissions:

- Read: Get role definition
- Write: Create or update custom role definition
- Read: Get role assignment
- Write: Create role assignment

The following built-in roles have the required Microsoft Authorization permissions:

- Owner
- User Access Administrator

To determine whether your user account role has the required permissions:

1. In the Azure portal, select **Resource groups**.
2. Select the resource group where the storage account is located, and then select **Access control (IAM)**.
3. Select the **Role assignments** tab.
4. Select the **Role** (for example, Owner or Contributor) for your user account.
5. In the **Resource Provider** list, select **Microsoft Authorization**.
  - **Role assignment** should have **Read** and **Write** permissions.
  - **Role definition** should have **Read** and **Write** permissions.

**Server endpoint creation fails, with this error: "MgmtServerJobFailed" (Error code: -2134375898 or 0x80c80226)**

This error occurs if the server endpoint path is on the system volume and cloud tiering is enabled. Cloud tiering is not supported on the system volume. To create a server endpoint on the system volume, disable cloud tiering when creating the server endpoint.

**Server endpoint creation fails, with this error: "MgmtServerJobFailed" (Error code: -2147024894 or 0x80070002)**

This error occurs if the server endpoint path specified is not valid. Verify the server endpoint path specified is a locally attached NTFS volume. Note, Azure File Sync does not support mapped drives as a server endpoint path.

**Server endpoint creation fails, with this error: "MgmtServerJobFailed" (Error code: -2134375640 or 0x80c80328)**

This error occurs if the server endpoint path specified is not an NTFS volume. Verify the server endpoint path specified is a locally attached NTFS volume. Note, Azure File Sync does not support mapped drives as a server endpoint path.

**Server endpoint creation fails, with this error: "MgmtServerJobFailed" (Error code: -2134347507 or**

## **0x80c8710d)**

This error occurs because Azure File Sync does not support server endpoints on volumes which have a compressed System Volume Information folder. To resolve this issue, decompress the System Volume Information folder. If the System Volume Information folder is the only folder compressed on the volume, perform the following steps:

1. Download [Psexec](#) tool.
2. Run the following command from an elevated command prompt to launch a command prompt running under the system account: **Psexec.exe -i -s -d cmd**
3. From the command prompt running under the system account, type the following commands and hit enter:  
**cd /d "drive letter:\System Volume Information"**  
**compact /u /s**

## **Server endpoint creation fails, with this error: "MgmtServerJobFailed" (Error code: -2134376345 or 0x80C80067)**

This error occurs if the server endpoints per server limit is reached. Azure File Sync currently supports up to 30 server endpoints per server. For more information, see [Azure File Sync scale targets](#).

## **Server endpoint creation fails, with this error: "MgmtServerJobFailed" (Error code: -2134376427 or 0x80c80015)**

This error occurs if another server endpoint is already syncing the server endpoint path specified. Azure File Sync does not support multiple server endpoints syncing the same directory or volume.

## **Server endpoint creation fails, with this error: "MgmtServerJobFailed" (Error code: -2160590967 or 0x80c80077)**

This error occurs if the server endpoint path contains orphaned tiered files. If a server endpoint was recently removed, wait until the orphaned tiered files cleanup has completed. An Event ID 6662 is logged to the Telemetry event log once the orphaned tiered files cleanup has started. An Event ID 6661 is logged once the orphaned tiered files cleanup has completed and a server endpoint can be recreated using the path. If the server endpoint creation fails after an Event ID 6661 is logged, remove the orphaned tiered files by performing the steps documented in the [Tiered files are not accessible on the server after deleting a server endpoint](#) section.

## **Server endpoint deletion fails, with this error: "MgmtServerJobExpired" (Error code: -2134347757 or 0x80c87013)**

This error occurs if the server is offline or doesn't have network connectivity. If the server is no longer available, unregister the server in the portal which will delete the server endpoints. To delete the server endpoints, follow the steps that are described in [Unregister a server with Azure File Sync](#).

## **Unable to open server endpoint properties page or update cloud tiering policy**

This issue can occur if a management operation on the server endpoint fails. If the server endpoint properties page does not open in the Azure portal, updating server endpoint using PowerShell commands from the server may fix this issue.

```
Get the server endpoint id based on the server endpoint DisplayName property
Get-AzStorageSyncServerEndpoint `-
 -ResourceGroupName myrgname `-
 -StorageSyncServiceName storagesvcname `-
 -SyncGroupName mysyncgroup | `-
 Tee-Object -Variable serverEndpoint

Update the free space percent policy for the server endpoint
Set-AzStorageSyncServerEndpoint `-
 -InputObject $serverEndpoint `-
 -CloudTiering `-
 -VolumeFreeSpacePercent 60
```

## **Server endpoint has a health status of "No Activity" or "Pending" and the server state on the**

## **registered servers blade is "Appears offline"**

This issue can occur if the Storage Sync Monitor process (AzureStorageSyncMonitor.exe) is not running or the server is unable to access the Azure File Sync service.

On the server that is showing as "Appears offline" in the portal, look at Event ID 9301 in the Telemetry event log (located under Applications and Services\Microsoft\FileSync\Agent in Event Viewer) to determine why the server is unable to access the Azure File Sync service.

- If **GetNextJob completed with status: 0** is logged, the server can communicate with the Azure File Sync service.
  - Open Task Manager on the server and verify the Storage Sync Monitor (AzureStorageSyncMonitor.exe) process is running. If the process is not running, first try restarting the server. If restarting the server does not resolve the issue, upgrade to the latest Azure File Sync [agent version](#).
- If **GetNextJob completed with status: -2134347756** is logged, the server is unable to communicate with the Azure File Sync service due to a firewall or proxy.
  - If the server is behind a firewall, verify port 443 outbound is allowed. If the firewall restricts traffic to specific domains, confirm the domains listed in the Firewall [documentation](#) are accessible.
  - If the server is behind a proxy, configure the machine-wide or app-specific proxy settings by following the steps in the Proxy [documentation](#).
  - Use the Test-StorageSyncNetworkConnectivity cmdlet to check network connectivity to the service endpoints. To learn more, see [Test network connectivity to service endpoints](#).
- If **GetNextJob completed with status: -2134347764** is logged, the server is unable to communicate with the Azure File Sync service due to an expired or deleted certificate.
  - Run the following PowerShell command on the server to reset the certificate used for authentication:

```
Reset-AzStorageSyncServerCertificate -ResourceGroupName <string> -StorageSyncServiceName <string>
```

## **Server endpoint has a health status of "No Activity" and the server state on the registered servers blade is "Online"**

A server endpoint health status of "No Activity" means the server endpoint has not logged sync activity in the past two hours.

To check current sync activity on a server, see [How do I monitor the progress of a current sync session?](#).

A server endpoint may not log sync activity for several hours due to a bug or insufficient system resources. Verify the latest Azure File Sync [agent version](#) is installed. If the issue persists, open a support request.

### **NOTE**

If the server state on the registered servers blade is "Appears Offline," perform the steps documented in the [Server endpoint has a health status of "No Activity" or "Pending" and the server state on the registered servers blade is "Appears offline"](#) section.

## **Sync**

### **If I created a file directly in my Azure file share over SMB or through the portal, how long does it take for the file to sync to servers in the sync group?**

Changes made to the Azure file share by using the Azure portal or SMB are not immediately detected and replicated like changes to the server endpoint. Azure Files does not yet have change notifications or journaling, so

there's no way to automatically initiate a sync session when files are changed. On Windows Server, Azure File Sync uses [Windows USN journaling](#) to automatically initiate a sync session when files change.

To detect changes to the Azure file share, Azure File Sync has a scheduled job called a *change detection job*. A change detection job enumerates every file in the file share, and then compares it to the sync version for that file. When the change detection job determines that files have changed, Azure File Sync initiates a sync session. The change detection job is initiated every 24 hours. Because the change detection job works by enumerating every file in the Azure file share, change detection takes longer in larger namespaces than in smaller namespaces. For large namespaces, it might take longer than once every 24 hours to determine which files have changed.

To immediately sync files that are changed in the Azure file share, the [Invoke-AzStorageSyncChangeDetection](#) PowerShell cmdlet can be used to manually initiate the detection of changes in the Azure file share. This cmdlet is intended for scenarios where some type of automated process is making changes in the Azure file share or the changes are done by an administrator (like moving files and directories into the share). For end user changes, the recommendation is to install the Azure File Sync agent in an IaaS VM and have end users access the file share through the IaaS VM. This way all changes will quickly sync to other agents without the need to use the `Invoke-AzStorageSyncChangeDetection` cmdlet. To learn more, see the [Invoke-AzStorageSyncChangeDetection](#) documentation.

#### NOTE

Changes made to an Azure file share using REST does not update the SMB last modified time and will not be seen as a change by sync.

We are exploring adding change detection for an Azure file share similar to USN for volumes on Windows Server. Help us prioritize this feature for future development by voting for it at [Azure Files UserVoice](#).

#### Server endpoint health is in a pending state for several hours

This issue is expected if you create a cloud endpoint and use an Azure file share that contains data. The change enumeration job that scans for changes in the Azure file share must complete before files can sync between the cloud and server endpoints. The time to complete the job is dependent on the size of the namespace in the Azure file share. The server endpoint health should update once the change enumeration job completes.

#### How do I monitor sync health?

- [Portal](#)
- [Server](#)

Within each sync group, you can drill down into its individual server endpoints to see the status of the last completed sync sessions. A green Health column and a Files Not Syncing value of 0 indicate that sync is working as expected. If this is not the case, see below for a list of common sync errors and how to handle files that are not syncing.

## How do I monitor the progress of a current sync session?

- [Portal](#)
- [Server](#)

Within your sync group, go to the server endpoint in question and look at the Sync Activity section to see the count of files uploaded or downloaded in the current sync session. Note that this status will be delayed by about 5 minutes, and if your sync session is small enough to be completed within this period, it may not be reported in the portal.

## How do I know if my servers are in sync with each other?

- [Portal](#)
- [Server](#)

For each server in a given sync group, make sure:

- The timestamps for the Last Attempted Sync for both upload and download are recent.
- The status is green for both upload and download.
- The Sync Activity field shows very few or no files remaining to sync.
- The Files Not Syncing field is 0 for both upload and download.

## How do I see if there are specific files or folders that are not syncing?

If your PerItemErrorCode on the server or Files Not Syncing count in the portal are greater than 0 for any given sync session, that means some items are failing to sync. Files and folders can have characteristics that prevent them from syncing. These characteristics can be persistent and require explicit action to resume sync, for example removing unsupported characters from the file or folder name. They can also be transient, meaning the file or folder will automatically resume sync; for example, files with open handles will automatically resume sync when the file is closed. When the Azure File Sync engine detects such a problem, an error log is produced that can be parsed to list the items currently not syncing properly.

To see these errors, run the **FileSyncErrorsReport.ps1** PowerShell script (located in the agent installation directory of the Azure File Sync agent) to identify files that failed to sync because of open handles, unsupported characters, or other issues. The ItemPath field tells you the location of the file in relation to the root sync directory. See the list of common sync errors below for remediation steps.

## NOTE

If the `FileSyncErrorsReport.ps1` script returns "There were no file errors found" or does not list per-item errors for the sync group, the cause is either:

- Cause 1: The last completed sync session did not have per-item errors. The portal should be updated soon to show 0 Files Not Syncing.
  - Check the [Event ID 9102](#) in the Telemetry event log to confirm the `PerItemErrorCode` is 0.
- Cause 2: The ItemResults event log on the server wrapped due to too many per-item errors and the event log no longer contains errors for this sync group.
  - To prevent this issue, increase the ItemResults event log size. The ItemResults event log can be found under "Applications and Services Logs\Microsoft\FileSync\Agent" in Event Viewer.

## Troubleshooting per file/directory sync errors

### ItemResults log - per-item sync errors

| HRESULT    | HRESULT (Decimal) | Error String                                  | Issue                                                                                                                                                                                   | Remediation                                                                                                                                                                          |
|------------|-------------------|-----------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x80070043 | -2147942467       | ERROR_BAD_NET_NAME                            | The tiered file on the server is not accessible. This issue occurs if the tiered file was not recalled prior to deleting a server endpoint.                                             | To resolve this issue, see <a href="#">Tiered files are not accessible on the server after deleting a server endpoint</a> .                                                          |
| 0x80c80207 | -2134375929       | ECS_E_SYNC_CONSTRAINT_CONFLICT                | The file or directory change cannot be synced yet because a dependent folder is not yet synced. This item will sync after the dependent changes are synced.                             | No action required. If the error persists for several days, use the <code>FileSyncErrorsReport.ps1</code> PowerShell script to determine why the dependent folder is not yet synced. |
| 0x80c80284 | -2134375804       | ECS_E_SYNC_CONSTRAINT_CONFLICT_SESSION_FAILED | The file or directory change cannot be synced yet because a dependent folder is not yet synced and the sync session failed. This item will sync after the dependent changes are synced. | No action required. If the error persists, investigate the sync session failure.                                                                                                     |
| 0x8007007b | -2147024773       | ERROR_INVALID_NAME                            | The file or directory name is invalid.                                                                                                                                                  | Rename the file or directory in question. See <a href="#">Handling unsupported characters</a> for more information.                                                                  |

| HRESULT    | HRESULT (Decimal) | Error String                     | Issue                                                                                                                                                                                           | Remediation                                                                                                                                                                                                                                   |
|------------|-------------------|----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x80c80255 | -2134375851       | ECS_E_XSMB_REST_IN_COMPATIBILITY | The file or directory name is invalid.                                                                                                                                                          | Rename the file or directory in question. See <a href="#">Handling unsupported characters</a> for more information.                                                                                                                           |
| 0x80c80018 | -2134376424       | ECS_E_SYNC_FILE_IN_USE           | The file cannot be synced because it's in use. The file will be synced when it's no longer in use.                                                                                              | No action required. Azure File Sync creates a temporary VSS snapshot once a day on the server to sync files that have open handles.                                                                                                           |
| 0x80c8031d | -2134375651       | ECS_E_CONCURRENCY_CHECK_FAILED   | The file has changed, but the change has not yet been detected by sync. Sync will recover after this change is detected.                                                                        | No action required.                                                                                                                                                                                                                           |
| 0x80070002 | -2147024894       | ERROR_FILE_NOT_FOUND             | The file was deleted and sync is not aware of the change.                                                                                                                                       | No action required. Sync will stop logging this error once change detection detects the file was deleted.                                                                                                                                     |
| 0x80070003 | -2147942403       | ERROR_PATH_NOT_FOUND             | Deletion of a file or directory cannot be synced because the item was already deleted in the destination and sync is not aware of the change.                                                   | No action required. Sync will stop logging this error once change detection runs on the destination and sync detects the item was deleted.                                                                                                    |
| 0x80c80205 | -2134375931       | ECS_E_SYNC_ITEM_SKIP             | The file or directory was skipped but will be synced during the next sync session. If this error is reported when downloading the item, the file or directory name is more than likely invalid. | No action required if this error is reported when uploading the file. If the error is reported when downloading the file, rename the file or directory in question. See <a href="#">Handling unsupported characters</a> for more information. |

| HRESULT    | HRESULT (Decimal) | Error String                                 | Issue                                                                                                                                                                      | Remediation                                                                                                                                                                                                  |
|------------|-------------------|----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x800700B7 | -2147024713       | ERROR_ALREADY_EXISTS                         | Creation of a file or directory cannot be synced because the item already exists in the destination and sync is not aware of the change.                                   | No action required. Sync will stop logging this error once change detection runs on the destination and sync is aware of this new item.                                                                      |
| 0x80c8603e | -2134351810       | ECS_E_AZURE_STORAGE_SHARE_SIZE_LIMIT_REACHED | The file cannot be synced because the Azure file share limit is reached.                                                                                                   | To resolve this issue, see <a href="#">You reached the Azure file share storage limit</a> section in the troubleshooting guide.                                                                              |
| 0x80c8027C | -2134375812       | ECS_E_ACCESS_DENIED_EFS                      | The file is encrypted by an unsupported solution (like NTFS EFS).                                                                                                          | Decrypt the file and use a supported encryption solution. For a list of support solutions, see <a href="#">Encryption solutions</a> section in the planning guide.                                           |
| 0x80c80283 | -2160591491       | ECS_E_ACCESS_DENIED_DFSRRO                   | The file is located on a DFS-R read-only replication folder.                                                                                                               | File is located on a DFS-R read-only replication folder. Azure Files Sync does not support server endpoints on DFS-R read-only replication folders. See <a href="#">planning guide</a> for more information. |
| 0x80070005 | -2147024891       | ERROR_ACCESS_DENIED                          | The file has a delete pending state.                                                                                                                                       | No action required. File will be deleted once all open file handles are closed.                                                                                                                              |
| 0x80c86044 | -2134351804       | ECS_E_AZURE_AUTHORIZATION_FAILED             | The file cannot be synced because the firewall and virtual network settings on the storage account are enabled and the server does not have access to the storage account. | Add the Server IP address or virtual network by following the steps documented in the <a href="#">Configure firewall and virtual network settings</a> section in the deployment guide.                       |
| 0x80c80243 | -2134375869       | ECS_E_SECURITY_DESCRIPTOR_SIZE_TOO_LARGE     | The file cannot be synced because the security descriptor size exceeds the 64 KiB limit.                                                                                   | To resolve this issue, remove access control entries (ACE) on the file to reduce the security descriptor size.                                                                                               |

| HRESULT    | HRESULT (Decimal) | Error String                    | Issue                                                                                                                                                                                                                        | Remediation                                                                                                                                                                                                                                                                         |
|------------|-------------------|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x8000ffff | -2147418113       | E_UNEXPECTED                    | The file cannot be synced due to an unexpected error.                                                                                                                                                                        | If the error persists for several days, please open a support case.                                                                                                                                                                                                                 |
| 0x80070020 | -2147024864       | ERROR_SHARING_VIOLATION         | The file cannot be synced because it's in use. The file will be synced when it's no longer in use.                                                                                                                           | No action required.                                                                                                                                                                                                                                                                 |
| 0x80c80017 | -2134376425       | ECS_E_SYNC_OPLOCK_BROKEN        | The file was changed during sync, so it needs to be synced again.                                                                                                                                                            | No action required.                                                                                                                                                                                                                                                                 |
| 0x80070017 | -2147024873       | ERROR_CRC                       | The file cannot be synced due to CRC error. This error can occur if a tiered file was not recalled prior to deleting a server endpoint or if the file is corrupt.                                                            | To resolve this issue, see <a href="#">Tiered files are not accessible on the server after deleting a server endpoint</a> to remove tiered files that are orphaned. If the error continues to occur after removing oprhaned tiered files, run <a href="#">chkdsk</a> on the volume. |
| 0x80c80200 | -2134375936       | ECS_E_SYNC_CONFLICT_NAME_EXISTS | The file cannot be synced because the maximum number of conflict files has been reached. Azure File Sync supports 100 conflict files per file. To learn more about file conflicts, see <a href="#">Azure File Sync FAQ</a> . | To resolve this issue, reduce the number of conflict files. The file will sync once the number of conflict files is less than 100.                                                                                                                                                  |

#### Handling unsupported characters

If the [FileSyncErrorsReport.ps1](#) PowerShell script shows failures due to unsupported characters (error code 0x8007007b or 0x80c80255), you should remove or rename the characters at fault from the respective file names. PowerShell will likely print these characters as question marks or empty rectangles since most of these characters have no standard visual encoding. The [Evaluation Tool](#) can be used to identify characters that are not supported.

The table below contains all of the unicode characters Azure File Sync does not yet support.

| CHARACTER SET                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | CHARACTER COUNT |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| <ul style="list-style-type: none"> <li>• 0x0000009D (osc operating system command)</li> <li>• 0x00000090 (dcs device control string)</li> <li>• 0x0000008F (ss3 single shift three)</li> <li>• 0x00000081 (high octet preset)</li> <li>• 0x0000007F (del delete)</li> <li>• 0x0000008D (ri reverse line feed)</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | 6               |
| 0x0000FDD0 - 0x0000FDEF (Arabic presentation forms-a)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              | 32              |
| 0x0000FFF0 - 0x0000FFFF (specials)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | 16              |
| <ul style="list-style-type: none"> <li>• 0x0001FFFE - 0x0001FFFF = 2 (noncharacter)</li> <li>• 0x0002FFFE - 0x0002FFFF = 2 (noncharacter)</li> <li>• 0x0003FFFE - 0x0003FFFF = 2 (noncharacter)</li> <li>• 0x0004FFFE - 0x0004FFFF = 2 (noncharacter)</li> <li>• 0x0005FFFE - 0x0005FFFF = 2 (noncharacter)</li> <li>• 0x0006FFFE - 0x0006FFFF = 2 (noncharacter)</li> <li>• 0x0007FFFE - 0x0007FFFF = 2 (noncharacter)</li> <li>• 0x0008FFFE - 0x0008FFFF = 2 (noncharacter)</li> <li>• 0x0009FFFE - 0x0009FFFF = 2 (noncharacter)</li> <li>• 0x000AFFFE - 0x000AFFFF = 2 (noncharacter)</li> <li>• 0x000BFFFE - 0x000BFFFF = 2 (noncharacter)</li> <li>• 0x000CFFFE - 0x000CFFFF = 2 (noncharacter)</li> <li>• 0x000DFFFE - 0x000DFFFF = 2 (noncharacter)</li> <li>• 0x000EFFFE - 0x000EFFFF = 2 (undefined)</li> <li>• 0x000FFFFE - 0x000FFFFFF = 2 (supplementary private use area)</li> </ul> | 30              |
| 0x0010FFFE, 0x0010FFFF                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | 2               |

### Common sync errors

**The sync session was cancelled.**

|                             |                 |
|-----------------------------|-----------------|
| <b>HRESULT</b>              | 0x800704c7      |
| <b>HRESULT (decimal)</b>    | -2147023673     |
| <b>Error string</b>         | ERROR_CANCELLED |
| <b>Remediation required</b> | No              |

Sync sessions may fail for various reasons including the server being restarted or updated, VSS snapshots, etc. Although this error looks like it requires follow-up, it is safe to ignore this error unless it persists over a period of several hours.

**A connection with the service could not be established.**

|                |            |
|----------------|------------|
| <b>HRESULT</b> | 0x80072ee7 |
|----------------|------------|

|                             |                             |
|-----------------------------|-----------------------------|
| <b>HRESULT (decimal)</b>    | -2147012889                 |
| <b>Error string</b>         | WININET_E_NAME_NOT_RESOLVED |
| <b>Remediation required</b> | Yes                         |

This error can occur whenever the Azure File Sync service is inaccessible from the server. You can troubleshoot this error by working through the following steps:

1. Verify the Windows service `FileSyncSvc.exe` is not blocked by your firewall.
2. Verify that port 443 is open to outgoing connections to the Azure File Sync service. You can do this with the `Test-NetConnection` cmdlet. The URL for the `<azure-file-sync-endpoint>` placeholder below can found in the [Azure File Sync proxy and firewall settings](#) document.

```
Test-NetConnection -ComputerName <azure-file-sync-endpoint> -Port 443
```

3. Ensure that the proxy configuration is set as anticipated. This can be done with the `Get-StorageSyncProxyConfiguration` cmdlet. More information on configuring the proxy configuration for Azure File Sync can be found in the [Azure File Sync proxy and firewall settings](#).

```
$agentPath = "C:\Program Files\Azure\StorageSyncAgent"
Import-Module "$agentPath\StorageSync.Management.ServerCmdlets.dll"
Get-StorageSyncProxyConfiguration
```

4. Use the `Test-StorageSyncNetworkConnectivity` cmdlet to check network connectivity to the service endpoints. To learn more, see [Test network connectivity to service endpoints](#).
5. Contact your network administrator for additional assistance troubleshooting network connectivity.

### The user request was throttled by the service.

|                             |                              |
|-----------------------------|------------------------------|
| <b>HRESULT</b>              | 0x80c8004c                   |
| <b>HRESULT (decimal)</b>    | -2134376372                  |
| <b>Error string</b>         | ECS_E_USER_REQUEST_THROTTLED |
| <b>Remediation required</b> | No                           |

No action is required; the server will try again. If this error persists for several hours, create a support request.

### Sync is blocked until change detection completes post restore

|                          |                                                     |
|--------------------------|-----------------------------------------------------|
| <b>HRESULT</b>           | 0x80c83075                                          |
| <b>HRESULT (decimal)</b> | -2134364043                                         |
| <b>Error string</b>      | ECS_E_SYNC_BLOCKED_ON_CHANGE_DETECTION_POST_RESTORE |

|                             |    |
|-----------------------------|----|
| <b>Remediation required</b> | No |
|-----------------------------|----|

No action is required. When a file or file share (cloud endpoint) is restored using Azure Backup, sync is blocked until change detection completes on the Azure file share. Change detection runs immediately once the restore is complete and the duration is based on the number of files in the file share.

#### **Sync failed because the sync database was unloaded.**

|                             |                                   |
|-----------------------------|-----------------------------------|
| <b>HRESULT</b>              | 0x80041295                        |
| <b>HRESULT (decimal)</b>    | -2147216747                       |
| <b>Error string</b>         | SYNC_E_METADATA_INVALID_OPERATION |
| <b>Remediation required</b> | No                                |

This error typically occurs when a backup application creates a VSS snapshot and the sync database is unloaded. If this error persists for several hours, create a support request.

#### **Sync can't access the Azure file share specified in the cloud endpoint.**

|                             |                                                     |
|-----------------------------|-----------------------------------------------------|
| <b>HRESULT</b>              | 0x80c8305f                                          |
| <b>HRESULT (decimal)</b>    | -2134364065                                         |
| <b>Error string</b>         | ECS_E_EXTERNAL_STORAGE_ACCOUNT_AUTHORIZATION_FAILED |
| <b>Remediation required</b> | Yes                                                 |

This error occurs because the Azure File Sync agent cannot access the Azure file share, which may be because the Azure file share or the storage account hosting it no longer exists. You can troubleshoot this error by working through the following steps:

1. [Verify the storage account exists.](#)
2. [Ensure the Azure file share exists.](#)
3. [Ensure Azure File Sync has access to the storage account.](#)
4. [Verify the firewall and virtual network settings on the storage account are configured properly \(if enabled\)](#)

#### **Sync failed because the request is not authorized to perform this operation.**

|                             |                                  |
|-----------------------------|----------------------------------|
| <b>HRESULT</b>              | 0x80c86044                       |
| <b>HRESULT (decimal)</b>    | -2134351804                      |
| <b>Error string</b>         | ECS_E_AZURE_AUTHORIZATION_FAILED |
| <b>Remediation required</b> | Yes                              |

This error occurs because the Azure File Sync agent is not authorized to access the Azure file share. You can troubleshoot this error by working through the following steps:

1. [Verify the storage account exists.](#)
2. [Ensure the Azure file share exists.](#)
3. [Verify the firewall and virtual network settings on the storage account are configured properly \(if enabled\)](#)
4. [Ensure Azure File Sync has access to the storage account.](#)

#### **The storage account name used could not be resolved.**

|                             |                                       |
|-----------------------------|---------------------------------------|
| <b>HRESULT</b>              | 0x80C83060                            |
| <b>HRESULT (decimal)</b>    | -2134364064                           |
| <b>Error string</b>         | ECS_E_STORAGE_ACCOUNT_NAME_UNRESOLVED |
| <b>Remediation required</b> | Yes                                   |

1. Check that you can resolve the storage DNS name from the server.

```
Test-NetConnection -ComputerName <storage-account-name>.file.core.windows.net -Port 443
```

2. [Verify the storage account exists.](#)
3. [Verify the firewall and virtual network settings on the storage account are configured properly \(if enabled\)](#)

#### **An unknown error occurred while accessing the storage account.**

|                             |                                     |
|-----------------------------|-------------------------------------|
| <b>HRESULT</b>              | 0x80c8308a                          |
| <b>HRESULT (decimal)</b>    | -2134364022                         |
| <b>Error string</b>         | ECS_E_STORAGE_ACCOUNT_UNKNOWN_ERROR |
| <b>Remediation required</b> | Yes                                 |

1. [Verify the storage account exists.](#)
2. [Verify the firewall and virtual network settings on the storage account are configured properly \(if enabled\)](#)

#### **Sync failed due to storage account locked.**

|                             |                              |
|-----------------------------|------------------------------|
| <b>HRESULT</b>              | 0x80c83092                   |
| <b>HRESULT (decimal)</b>    | -2134364014                  |
| <b>Error string</b>         | ECS_E_STORAGE_ACCOUNT_LOCKED |
| <b>Remediation required</b> | Yes                          |

This error occurs because the storage account has a read-only [resource lock](#). To resolve this issue, remove the

read-only resource lock on the storage account.

**Sync failed due to a problem with the sync database.**

|                             |                      |
|-----------------------------|----------------------|
| <b>HRESULT</b>              | 0x8e5e044e           |
| <b>HRESULT (decimal)</b>    | -1906441138          |
| <b>Error string</b>         | JET_errWriteConflict |
| <b>Remediation required</b> | Yes                  |

This error occurs when there is a problem with the internal database used by Azure File Sync. When this issue occurs, create a support request and we will contact you to help you resolve this issue.

**The Azure File Sync agent version installed on the server is not supported.**

|                             |                             |
|-----------------------------|-----------------------------|
| <b>HRESULT</b>              | 0x80C8306B                  |
| <b>HRESULT (decimal)</b>    | -2134364053                 |
| <b>Error string</b>         | ECS_E_AGENT_VERSION_BLOCKED |
| <b>Remediation required</b> | Yes                         |

This error occurs if the Azure File Sync agent version installed on the server is not supported. To resolve this issue, [upgrade](#) to a [supported agent version](#).

**You reached the Azure file share storage limit.**

|                             |                                              |
|-----------------------------|----------------------------------------------|
| <b>HRESULT</b>              | 0x80c8603e                                   |
| <b>HRESULT (decimal)</b>    | -2134351810                                  |
| <b>Error string</b>         | ECS_E_AZURE_STORAGE_SHARE_SIZE_LIMIT_REACHED |
| <b>Remediation required</b> | Yes                                          |

This error occurs when the Azure file share storage limit has been reached, which can happen if a quota is applied for an Azure file share or if the usage exceeds the limits for an Azure file share. For more information, see the [current limits for an Azure file share](#).

1. Navigate to the sync group within the Storage Sync Service.
2. Select the cloud endpoint within the sync group.
3. Note the Azure file share name in the opened pane.
4. Select the linked storage account. If this link fails, the referenced storage account has been removed.

Cloud Endpoint Properties

finalvideos - PREVIEW

Refresh Delete

|                    |                 |
|--------------------|-----------------|
| Provisioning State |                 |
| Resource ID        | /subscriptions/ |
| Connected Storage  |                 |
| Azure File Share   | finalvideos     |
| Storage Account    | serversummitafs |
| Resource Group     | ServerSummit    |
| Subscription       |                 |

5. Select **Files** to view the list of file shares.
6. Click the three dots at the end of the row for the Azure file share referenced by the cloud endpoint.
7. Verify that the **Usage** is below the **Quota**. Note unless an alternate quota has been specified, the quota will match the [maximum size of the Azure file share](#).

Share properties

finalvideos

|                      |                                                                                                             |
|----------------------|-------------------------------------------------------------------------------------------------------------|
| <b>NAME</b>          | finalvideos                                                                                                 |
| <b>URL</b>           | <a href="https://serversummitafs.file.core.windows.net/">https://serversummitafs.file.core.windows.net/</a> |
| <b>LAST MODIFIED</b> | 7/8/2018, 12:02:27 PM                                                                                       |
| <b>ETAG</b>          |                                                                                                             |
| <b>QUOTA</b>         | 5 TiB                                                                                                       |
| <b>USAGE</b>         | 2 GiB                                                                                                       |

If the share is full and a quota is not set, one possible way of fixing this issue is to make each subfolder of the current server endpoint into its own server endpoint in their own separate sync groups. This way each subfolder will sync to individual Azure file shares.

#### The Azure file share cannot be found.

|                |            |
|----------------|------------|
| <b>HRESULT</b> | 0x80c86030 |
|----------------|------------|

|                             |                                  |
|-----------------------------|----------------------------------|
| <b>HRESULT (decimal)</b>    | -2134351824                      |
| <b>Error string</b>         | ECS_E_AZURE_FILE_SHARE_NOT_FOUND |
| <b>Remediation required</b> | Yes                              |

This error occurs when the Azure file share is not accessible. To troubleshoot:

1. [Verify the storage account exists.](#)
2. [Ensure the Azure file share exists.](#)

If the Azure file share was deleted, you need to create a new file share and then recreate the sync group.

#### **Sync is paused while this Azure subscription is suspended.**

|                             |                                              |
|-----------------------------|----------------------------------------------|
| <b>HRESULT</b>              | 0x80C83076                                   |
| <b>HRESULT (decimal)</b>    | -2134364042                                  |
| <b>Error string</b>         | ECS_E_SYNC_BLOCKED_ON_SUSPENDED_SUBSCRIPTION |
| <b>Remediation required</b> | Yes                                          |

This error occurs when the Azure subscription is suspended. Sync will be reenabled when the Azure subscription is restored. See [Why is my Azure subscription disabled and how do I reactivate it?](#) for more information.

#### **The storage account has a firewall or virtual networks configured.**

|                             |                                    |
|-----------------------------|------------------------------------|
| <b>HRESULT</b>              | 0x80c8306c                         |
| <b>HRESULT (decimal)</b>    | -2134364052                        |
| <b>Error string</b>         | ECS_E_MGMT_STORAGEACLSNOTSUPPORTED |
| <b>Remediation required</b> | Yes                                |

This error occurs when the Azure file share is inaccessible because of a storage account firewall or because the storage account belongs to a virtual network. Verify the firewall and virtual network settings on the storage account are configured properly. For more information, see [Configure firewall and virtual network settings](#).

#### **Sync failed due to a problem with the sync database.**

|                             |                                        |
|-----------------------------|----------------------------------------|
| <b>HRESULT</b>              | 0x80c80219                             |
| <b>HRESULT (decimal)</b>    | -2134375911                            |
| <b>Error string</b>         | ECS_E_SYNC_METADATA_WRITE_LOCK_TIMEOUT |
| <b>Remediation required</b> | No                                     |

This error usually resolves itself, and can occur if there are:

- A high number of file changes across the servers in the sync group.
- A large number of errors on individual files and directories.

If this error persists for longer than a few hours, create a support request and we will contact you to help you resolve this issue.

**The server failed to establish a secure connection. The cloud service received an unexpected certificate.**

|                             |                      |
|-----------------------------|----------------------|
| <b>HRESULT</b>              | 0x800b0109           |
| <b>HRESULT (decimal)</b>    | -2146762487          |
| <b>Error string</b>         | CERT_E_UNTRUSTEDROOT |
| <b>Remediation required</b> | Yes                  |

This error can happen if your organization is using an SSL terminating proxy or if a malicious entity is intercepting the traffic between your server and the Azure File Sync service. If you are certain that this is expected (because your organization is using an SSL terminating proxy), you skip certificate verification with a registry override.

1. Create the SkipVerifyingPinnedRootCertificate registry value.

```
New-ItemProperty -Path HKLM:\SOFTWARE\Microsoft\Azure\StorageSync -Name
SkipVerifyingPinnedRootCertificate -PropertyType DWORD -Value 1
```

2. Restart the sync service on the registered server.

```
Restart-Service -Name FileSyncSvc -Force
```

By setting this registry value, the Azure File Sync agent will accept any locally trusted SSL certificate when transferring data between the server and the cloud service.

**A connection with the service could not be established.**

|                             |                   |
|-----------------------------|-------------------|
| <b>HRESULT</b>              | 0x80072ee2        |
| <b>HRESULT (decimal)</b>    | -2147012894       |
| <b>Error string</b>         | WININET_E_TIMEOUT |
| <b>Remediation required</b> | Yes               |

This error can occur whenever the Azure File Sync service is inaccessible from the server. You can troubleshoot this error by working through the following steps:

1. Verify the Windows service `FileSyncSvc.exe` is not blocked by your firewall.
2. Verify that port 443 is open to outgoing connections to the Azure File Sync service. You can do this with

the `Test-NetConnection` cmdlet. The URL for the `<azure-file-sync-endpoint>` placeholder below can found in the [Azure File Sync proxy and firewall settings](#) document.

```
Test-NetConnection -ComputerName <azure-file-sync-endpoint> -Port 443
```

3. Ensure that the proxy configuration is set as anticipated. This can be done with the `Get-StorageSyncProxyConfiguration` cmdlet. More information on configuring the proxy configuration for Azure File Sync can be found in the [Azure File Sync proxy and firewall settings](#).

```
$agentPath = "C:\Program Files\Azure\StorageSyncAgent"
Import-Module "$agentPath\StorageSync.Management.ServerCmdlets.dll"
Get-StorageSyncProxyConfiguration
```

4. Use the `Test-StorageSyncNetworkConnectivity` cmdlet to check network connectivity to the service endpoints. To learn more, see [Test network connectivity to service endpoints](#).
5. Contact your network administrator for additional assistance troubleshooting network connectivity.

#### **Sync failed due to a problem with authentication.**

|                             |                                |
|-----------------------------|--------------------------------|
| <b>HRESULT</b>              | 0x80c80300                     |
| <b>HRESULT (decimal)</b>    | -2134375680                    |
| <b>Error string</b>         | ECS_E_SERVER_CREDENTIAL_NEEDED |
| <b>Remediation required</b> | Yes                            |

This error typically occurs because the server time is incorrect. If the server is running in a virtual machine, verify the time on the host is correct.

#### **Sync failed due to certificate expiration.**

|                             |                             |
|-----------------------------|-----------------------------|
| <b>HRESULT</b>              | 0x80c83078                  |
| <b>HRESULT (decimal)</b>    | -2134364040                 |
| <b>Error string</b>         | ECS_E_AUTH_SRV_CERT_EXPIRED |
| <b>Remediation required</b> | Yes                         |

This error occurs because the certificate used for authentication is expired.

To confirm the certificate is expired, perform the following steps:

1. Open the Certificates MMC snap-in, select Computer Account and navigate to Certificates (Local Computer)\Personal\Certificates.
2. Check if the client authentication certificate is expired.

If the client authentication certificate is expired, perform the following steps to resolve the issue:

1. Verify Azure File Sync agent version 4.0.1.0 or later is installed.

2. Run the following PowerShell command on the server:

```
Reset-AzStorageSyncServerCertificate -ResourceGroupName <string> -StorageSyncServiceName <string>
```

#### Sync failed due to authentication certificate not found.

|                             |                               |
|-----------------------------|-------------------------------|
| <b>HRESULT</b>              | 0x80c80228                    |
| <b>HRESULT (decimal)</b>    | -2134375896                   |
| <b>Error string</b>         | ECS_E_AUTH_SRV_CERT_NOT_FOUND |
| <b>Remediation required</b> | Yes                           |

This error occurs because the certificate used for authentication is not found.

To resolve this issue, perform the following steps:

1. Verify Azure File Sync agent version 4.0.1.0 or later is installed.
2. Run the following PowerShell command on the server:

```
Reset-AzStorageSyncServerCertificate -ResourceGroupName <string> -StorageSyncServiceName <string>
```

#### Sync failed due to authentication identity not found.

|                             |                               |
|-----------------------------|-------------------------------|
| <b>HRESULT</b>              | 0x80c83079                    |
| <b>HRESULT (decimal)</b>    | -2134364039                   |
| <b>Error string</b>         | ECS_E_AUTH_IDENTITY_NOT_FOUND |
| <b>Remediation required</b> | Yes                           |

This error occurs because the server endpoint deletion failed and the endpoint is now in a partially deleted state.

To resolve this issue, retry deleting the server endpoint.

#### The volume where the server endpoint is located is low on disk space.

|                             |                    |
|-----------------------------|--------------------|
| <b>HRESULT</b>              | 0x8e5e0211         |
| <b>HRESULT (decimal)</b>    | -1906441711        |
| <b>Error string</b>         | JET_errLogDiskFull |
| <b>Remediation required</b> | Yes                |
| <b>HRESULT</b>              | 0x80c8031a         |

|                             |                                |
|-----------------------------|--------------------------------|
| <b>HRESULT (decimal)</b>    | -2134375654                    |
| <b>Error string</b>         | ECS_E_NOT_ENOUGH_LOCAL_STORAGE |
| <b>Remediation required</b> | Yes                            |

This error occurs because the volume has filled up. This error commonly occurs because files outside the server endpoint are using up space on the volume. Free up space on the volume by adding additional server endpoints, moving files to a different volume, or increasing the size of the volume the server endpoint is on.

#### **The service is not yet ready to sync with this server endpoint.**

|                             |                         |
|-----------------------------|-------------------------|
| <b>HRESULT</b>              | 0x80c8300f              |
| <b>HRESULT (decimal)</b>    | -2134364145             |
| <b>Error string</b>         | ECS_E_REPLICA_NOT_READY |
| <b>Remediation required</b> | No                      |

This error occurs because the cloud endpoint was created with content already existing on the Azure file share. Azure File Sync must scan the Azure file share for all content before allowing the server endpoint to proceed with its initial synchronization.

#### **Sync failed due to problems with many individual files.**

|                             |                                                  |
|-----------------------------|--------------------------------------------------|
| <b>HRESULT</b>              | 0x80c8023b                                       |
| <b>HRESULT (decimal)</b>    | -2134375877                                      |
| <b>Error string</b>         | ECS_E_SYNC_METADATA_KNOWLEDGE_SOFT_LIMIT_REACHED |
| <b>Remediation required</b> | Yes                                              |
| <b>HRESULT</b>              | 0x80c8021c                                       |
| <b>HRESULT (decimal)</b>    | -2134375908                                      |
| <b>Error string</b>         | ECS_E_SYNC_METADATA_KNOWLEDGE_LIMIT_REACHED      |
| <b>Remediation required</b> | Yes                                              |
| <b>HRESULT</b>              | 0x80c80253                                       |
| <b>HRESULT (decimal)</b>    | -2134375853                                      |

|                             |                                |
|-----------------------------|--------------------------------|
| <b>Error string</b>         | ECS_E_TOO_MANY_PER_ITEM_ERRORS |
| <b>Remediation required</b> | Yes                            |

In cases where there are many per file sync errors, sync sessions may begin to fail.

**NOTE**

Azure File Sync creates a temporary VSS snapshot once a day on the server to sync files that have open handles.

**Sync failed due to a problem with the server endpoint path.**

|                             |                         |
|-----------------------------|-------------------------|
| <b>HRESULT</b>              | 0x80c80019              |
| <b>HRESULT (decimal)</b>    | -2134376423             |
| <b>Error string</b>         | ECS_E_SYNC_INVALID_PATH |
| <b>Remediation required</b> | Yes                     |

Ensure the path exists, is on a local NTFS volume, and is not a reparse point or existing server endpoint.

**Sync failed because the filter driver version is not compatible with the agent version**

|                             |                                   |
|-----------------------------|-----------------------------------|
| <b>HRESULT</b>              | 0x80C80277                        |
| <b>HRESULT (decimal)</b>    | -2134375817                       |
| <b>Error string</b>         | ECS_E_INCOMPATIBLE_FILTER_VERSION |
| <b>Remediation required</b> | Yes                               |

This error occurs because the Cloud Tiering filter driver (StorageSync.sys) version loaded is not compatible with the Storage Sync Agent (FileSyncSvc) service. If the Azure File Sync agent was upgraded, restart the server to complete the installation. If the error continues to occur, uninstall the agent, restart the server and reinstall the Azure File Sync agent.

**The service is currently unavailable.**

|                             |                           |
|-----------------------------|---------------------------|
| <b>HRESULT</b>              | 0x80c8004b                |
| <b>HRESULT (decimal)</b>    | -2134376373               |
| <b>Error string</b>         | ECS_E_SERVICE_UNAVAILABLE |
| <b>Remediation required</b> | No                        |

This error occurs because the Azure File Sync service is unavailable. This error will auto-resolve when the Azure

File Sync service is available again.

#### **Sync failed due to an exception.**

|                             |                 |
|-----------------------------|-----------------|
| <b>HRESULT</b>              | 0x80131500      |
| <b>HRESULT (decimal)</b>    | -2146233088     |
| <b>Error string</b>         | COR_E_EXCEPTION |
| <b>Remediation required</b> | No              |

This error occurs because sync failed due to an exception. If the error persists for several hours, please create a support request.

#### **Sync failed because the storage account has failed over to another region.**

|                             |                                   |
|-----------------------------|-----------------------------------|
| <b>HRESULT</b>              | 0x80c83073                        |
| <b>HRESULT (decimal)</b>    | -2134364045                       |
| <b>Error string</b>         | ECS_E_STORAGE_ACCOUNT_FAILED_OVER |
| <b>Remediation required</b> | Yes                               |

This error occurs because the storage account has failed over to another region. Azure File Sync does not support the storage account failover feature. 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. To resolve this issue, move the storage account to the primary region.

#### **Sync failed due to a transient problem with the sync database.**

|                             |                                     |
|-----------------------------|-------------------------------------|
| <b>HRESULT</b>              | 0x80c8020e                          |
| <b>HRESULT (decimal)</b>    | -2134375922                         |
| <b>Error string</b>         | ECS_E_SYNC_METADATA_WRITELEASE_LOST |
| <b>Remediation required</b> | No                                  |

This error occurs because of an internal problem with the sync database. This error will auto-resolve when sync retries. If this error continues for an extend period of time, create a support request and we will contact you to help you resolve this issue.

#### **Sync failed due to change in Azure Active Directory tenant**

|                |            |
|----------------|------------|
| <b>HRESULT</b> | 0x80c83088 |
|----------------|------------|

|                             |                          |
|-----------------------------|--------------------------|
| <b>HRESULT (decimal)</b>    | -2134364024              |
| <b>Error string</b>         | ECS_E_INVALID_AAD_TENANT |
| <b>Remediation required</b> | Yes                      |

This error occurs because Azure File Sync does not currently support moving the subscription to a different Azure Active Directory tenant.

To resolve this issue, perform one of the following options:

- **Option 1 (recommended):** Move the subscription back to the original Azure Active Directory tenant
- **Option 2:** Delete and recreate the current sync group. If cloud tiering was enabled on the server endpoint, delete the sync group and then perform the steps documented in the [Cloud Tiering section](#) to remove the orphaned tiered files prior to recreating the sync groups.

#### **Sync failed due to firewall and virtual network exception not configured**

|                             |                                    |
|-----------------------------|------------------------------------|
| <b>HRESULT</b>              | 0x80c83096                         |
| <b>HRESULT (decimal)</b>    | -2134364010                        |
| <b>Error string</b>         | ECS_E_MGMT_STORAGEACLSBYPASSNOTSET |
| <b>Remediation required</b> | Yes                                |

This error occurs if the firewall and virtual network settings are enabled on the storage account and the "Allow trusted Microsoft services to access this storage account" exception is not checked. To resolve this issue, follow the steps documented in the [Configure firewall and virtual network settings](#) section in the deployment guide.

#### **Sync failed because permissions on the System Volume Information folder are incorrect.**

|                             |                     |
|-----------------------------|---------------------|
| <b>HRESULT</b>              | 0x80070005          |
| <b>HRESULT (decimal)</b>    | -2147024891         |
| <b>Error string</b>         | ERROR_ACCESS_DENIED |
| <b>Remediation required</b> | Yes                 |

This error can occur if the NT AUTHORITY\SYSTEM account does not have permissions to the System Volume Information folder on the volume where the server endpoint is located. Note, if individual files are failing to sync with ERROR\_ACCESS\_DENIED, perform the steps documented in the [Troubleshooting per file/directory sync errors](#) section.

To resolve this issue, perform the following steps:

1. Download [Psexec](#) tool.
2. Run the following command from an elevated command prompt to launch a command prompt using the system account: **PsExec.exe -i -s -d cmd**
3. From the command prompt running under the system account, run the following command to confirm the NT

AUTHORITY\SYSTEM account does not have access to the System Volume Information folder: **cacls "drive letter:\system volume information" /T /C**

4. If the NT AUTHORITY\SYSTEM account does not have access to the System Volume Information folder, run the following command: **cacls "drive letter:\system volume information" /T /E /G "NT AUTHORITY\SYSTEM:F"**
  - If step #4 fails with access denied, run the following command to take ownership of the System Volume Information folder and then repeat step #4: **takeown /A /R /F "drive letter:\System Volume Information"**

#### **Sync failed because the Azure file share was deleted and recreated.**

|                             |                                 |
|-----------------------------|---------------------------------|
|                             |                                 |
| <b>HRESULT</b>              | 0x80c8027e                      |
| <b>HRESULT (decimal)</b>    | -2134375810                     |
| <b>Error string</b>         | ECS_E_SYNC_REPLICA_ROOT_CHANGED |
| <b>Remediation required</b> | Yes                             |

This error occurs because Azure File Sync does not support deleting and recreating an Azure file share in the same sync group.

To resolve this issue, delete and recreate the sync group by performing the following steps:

1. Delete all server endpoints in the sync group.
2. Delete the cloud endpoint.
3. Delete the sync group.
4. If cloud tiering was enabled on a server endpoint, delete the orphaned tiered files on the server by performing the steps documented in the [Tiered files are not accessible on the server after deleting a server endpoint](#) section.
5. Recreate the sync group.

#### **Sync failed because the HTTP request was redirected**

|                             |                                  |
|-----------------------------|----------------------------------|
|                             |                                  |
| <b>HRESULT</b>              | 0x80190133                       |
| <b>HRESULT (decimal)</b>    | -2145844941                      |
| <b>Error string</b>         | HTTP_E_STATUS_REDIRECT_KEEP_VERB |
| <b>Remediation required</b> | Yes                              |

This error occurs because Azure File Sync does not support HTTP redirection (3xx status code). To resolve this issue, disable HTTP redirect on your proxy server or network device.

#### **A timeout occurred during offline data transfer, but it is still in progress.**

|                          |             |
|--------------------------|-------------|
|                          |             |
| <b>HRESULT</b>           | 0x80c83085  |
| <b>HRESULT (decimal)</b> | -2134364027 |

|                             |                                   |
|-----------------------------|-----------------------------------|
| <b>Error string</b>         | ECS_E_DATA_INGESTION_WAIT_TIMEOUT |
| <b>Remediation required</b> | No                                |

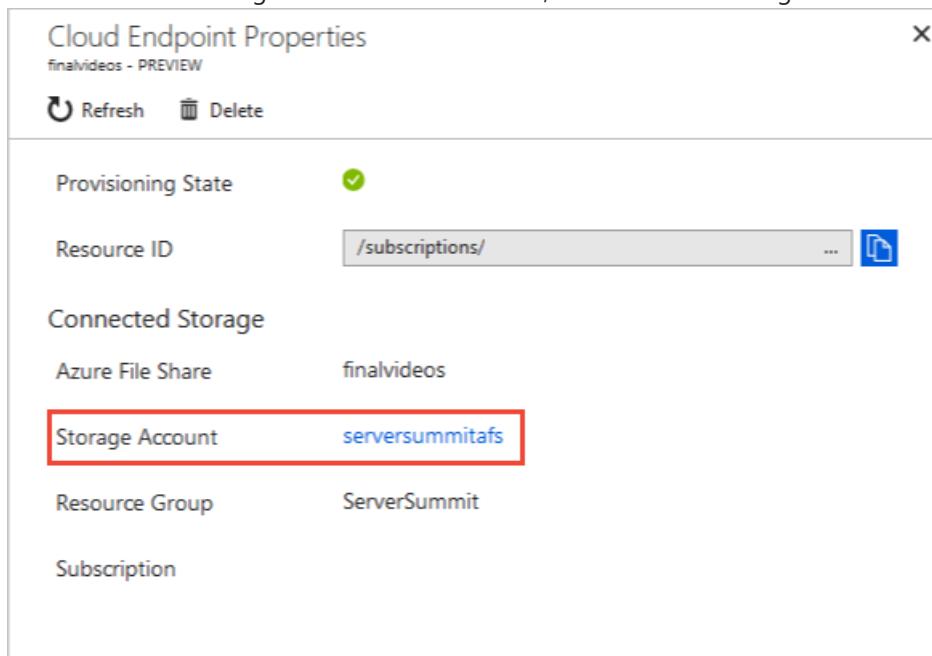
This error occurs when a data ingestion operation exceeds the timeout. This error can be ignored if sync is making progress (AppliedItemCount is greater than 0). See [How do I monitor the progress of a current sync session?](#)

### Common troubleshooting steps

#### Verify the storage account exists.

- [Portal](#)
- [PowerShell](#)

1. Navigate to the sync group within the Storage Sync Service.
2. Select the cloud endpoint within the sync group.
3. Note the Azure file share name in the opened pane.
4. Select the linked storage account. If this link fails, the referenced storage account has been removed.



#### Ensure the Azure file share exists.

- [Portal](#)
- [PowerShell](#)

1. Click **Overview** on the left-hand table of contents to return to the main storage account page.
2. Select **Files** to view the list of file shares.
3. Verify the file share referenced by the cloud endpoint appears in the list of file shares (you should have noted this in step 1 above).

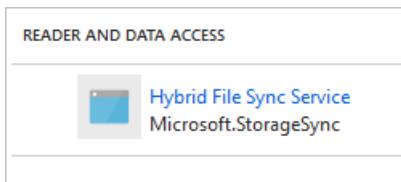
#### Ensure Azure File Sync has access to the storage account.

- [Portal](#)
- [PowerShell](#)

1. Click **Access control (IAM)** on the left-hand table of contents.
2. Click the **Role assignments** tab to list the users and applications (*service principals*) that have access

to your storage account.

3. Verify **Hybrid File Sync Service** appears in the list with the **Reader and Data Access** role.



If **Hybrid File Sync Service** does not appear in the list, perform the following steps:

- Click **Add**.
- In the **Role** field, select **Reader and Data Access**.
- In the **Select** field, type **Hybrid File Sync Service**, select the role and click **Save**.

#### How do I prevent users from creating files containing unsupported characters on the server?

You can use [File Server Resource Manager \(FSRM\) File Screens](#) to block files with unsupported characters in their names from being created on the server. You may have to do this using PowerShell as most of the unsupported characters are not printable and so you need to cast their hexadecimal representations as characters first.

First create an FSRM File Group using the [New-FsrmFileGroup cmdlet](#). This example defines the group to contain only two of the unsupported characters, but you can include as many of the characters as necessary in your file group.

```
New-FsrmFileGroup -Name "Unsupported characters" -IncludePattern @(("*" + [char]0x00000090 + "*"), ("*" + [char]0x0000008F + "*"))
```

Once you have defined an FSRM File Group, you can create an FSRM File Screen using the [New-FsrmFileScreen cmdlet](#).

```
New-FsrmFileScreen -Path "E:\AFSdataset" -Description "Filter unsupported characters" -IncludeGroup "Unsupported characters"
```

#### IMPORTANT

Note that file screens should only be used to block the creation of characters not supported by Azure File Sync. If file screens are used in other scenarios, sync will continually try to download the files from the Azure file share to the server and will be blocked due to the file screen, resulting in high data egress.

## Cloud tiering

There are two paths for failures in cloud tiering:

- Files can fail to tier, which means that Azure File Sync unsuccessfully attempts to tier a file to Azure Files.
- Files can fail to recall, which means that the Azure File Sync file system filter (StorageSync.sys) fails to download data when a user attempts to access a file which has been tiered.

There are two main classes of failures that can happen via either failure path:

- Cloud storage failures
  - *Transient storage service availability issues*. For more information, see the [Service Level Agreement \(SLA\) for Azure Storage](#).
  - *Inaccessible Azure file share*. This failure typically happens when you delete the Azure file share when it

is still a cloud endpoint in a sync group.

- *Inaccessible storage account.* This failure typically happens when you delete the storage account while it still has an Azure file share which is a cloud endpoint in a sync group.
- Server failures
  - *Azure File Sync file system filter (StorageSync.sys) is not loaded.* In order to respond to tiering/recall requests, the Azure File Sync file system filter must be loaded. The filter not being loaded can happen for several reasons, but the most common reason is that an administrator unloaded it manually. The Azure File Sync file system filter must be loaded at all times for Azure File Sync to properly function.
  - *Missing, corrupt, or otherwise broken reparse point.* A reparse point is a special data structure on a file that consists of two parts:
    1. A reparse tag, which indicates to the operating system that the Azure File Sync file system filter (StorageSync.sys) may need to do some action on IO to the file.
    2. Reparse data, which indicates to the file system filter the URI of the file on the associated cloud endpoint (the Azure file share).

The most common way a reparse point could become corrupted is if an administrator attempts to modify either the tag or its data.

- *Network connectivity issues.* In order to tier or recall a file, the server must have internet connectivity.

The following sections indicate how to troubleshoot cloud tiering issues and determine if an issue is a cloud storage issue or a server issue.

### **How to monitor tiering activity on a server**

To monitor tiering activity on a server, use Event ID 9003, 9016 and 9029 in the Telemetry event log (located under Applications and Services\Microsoft\FileSync\Agent in Event Viewer).

- Event ID 9003 provides error distribution for a server endpoint. For example, Total Error Count, ErrorCode, etc. Note, one event is logged per error code.
- Event ID 9016 provides ghosting results for a volume. For example, Free space percent is, Number of files ghosted in session, Number of files failed to ghost, etc.
- Event ID 9029 provides ghosting session information for a server endpoint. For example, Number of files attempted in the session, Number of files tiered in the session, Number of files already tiered, etc.

### **How to monitor recall activity on a server**

To monitor recall activity on a server, use Event ID 9005, 9006, 9009 and 9059 in the Telemetry event log (located under Applications and Services\Microsoft\FileSync\Agent in Event Viewer).

- Event ID 9005 provides recall reliability for a server endpoint. For example, Total unique files accessed, Total unique files with failed access, etc.
- Event ID 9006 provides recall error distribution for a server endpoint. For example, Total Failed Requests, ErrorCode, etc. Note, one event is logged per error code.
- Event ID 9009 provides recall session information for a server endpoint. For example, DurationSeconds, CountFilesRecallSucceeded, CountFilesRecallFailed, etc.
- Event ID 9059 provides application recall distribution for a server endpoint. For example, ShareId, Application Name, and TotalEgressNetworkBytes.

### **How to troubleshoot files that fail to tier**

If files fail to tier to Azure Files:

1. In Event Viewer, review the telemetry, operational and diagnostic event logs, located under Applications and Services\Microsoft\FileSync\Agent.
  - a. Verify the files exist in the Azure file share.

**NOTE**

A file must be synced to an Azure file share before it can be tiered.

- b. Verify the server has internet connectivity.
- c. Verify the Azure File Sync filter drivers (StorageSync.sys and StorageSyncGuard.sys) are running:
  - At an elevated command prompt, run `f1tmc`. Verify that the StorageSync.sys and StorageSyncGuard.sys file system filter drivers are listed.

**NOTE**

An Event ID 9003 is logged once an hour in the Telemetry event log if a file fails to tier (one event is logged per error code). Check the [Tiering errors and remediation](#) section to see if remediation steps are listed for the error code.

**Tiering errors and remediation**

| HRESULT    | HRESULT (DECIMAL) | ERROR STRING                    | ISSUE                                                                   | REMEDIATION                                                                                            |
|------------|-------------------|---------------------------------|-------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|
| 0x80c86043 | -2134351805       | ECS_E_GHOSTING_FILE_IN_USE      | The file failed to tier because it's in use.                            | No action required. The file will be tiered when it's no longer in use.                                |
| 0x80c80241 | -2134375871       | ECS_E_GHOSTING_EXCLUDED_BY_SYNC | The file failed to tier because it's excluded by sync.                  | No action required. Files in the sync exclusion list cannot be tiered.                                 |
| 0x80c86042 | -2134351806       | ECS_E_GHOSTING_FILE_NOT_FOUND   | The file failed to tier because it was not found on the server.         | No action required. If the error persists, check if the file exists on the server.                     |
| 0x80c83053 | -2134364077       | ECS_E_CREATE_SV_FILE_DELETED    | The file failed to tier because it was deleted in the Azure file share. | No action required. The file should be deleted on the server when the next download sync session runs. |
| 0x80c8600e | -2134351858       | ECS_E_AZURE_SERVER_BUSY         | The file failed to tier due to a network issue.                         | No action required. If the error persists, check network connectivity to the Azure file share.         |
| 0x80072ee7 | -2147012889       | WININET_E_NAME_NOT_RESOLVED     | The file failed to tier due to a network issue.                         | No action required. If the error persists, check network connectivity to the Azure file share.         |

| HRESULT    | HRESULT (DECIMAL) | ERROR STRING                   | ISSUE                                                                                                                                    | REMEDIATION                                                                                                                                                                                                                                                                                                       |
|------------|-------------------|--------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x80070005 | -2147024891       | ERROR_ACCESS_DENIED            | The file failed to tier due to access denied error. This error can occur if the file is located on a DFS-R read-only replication folder. | Azure Files Sync does not support server endpoints on DFS-R read-only replication folders. See <a href="#">planning guide</a> for more information.                                                                                                                                                               |
| 0x80072efe | -2147012866       | WININET_E_CONNECTION_ABORTED   | The file failed to tier due to a network issue.                                                                                          | No action required. If the error persists, check network connectivity to the Azure file share.                                                                                                                                                                                                                    |
| 0x80c80261 | -2134375839       | ECS_E_GHOSTING_MIN_FILE_SIZE   | The file failed to tier because the file size is less than the supported size.                                                           | If the agent version is less than 9.0, the minimum supported file size is 64kb. If agent version is 9.0 and newer, the minimum supported file size is based on the file system cluster size (double file system cluster size). For example, if the file system cluster size is 4kb, the minimum file size is 8kb. |
| 0x80c83007 | -2134364153       | ECS_E_STORAGE_ERROR            | The file failed to tier due to an Azure storage issue.                                                                                   | If the error persists, open a support request.                                                                                                                                                                                                                                                                    |
| 0x800703e3 | -2147023901       | ERROR_OPERATION_ABORTED        | The file failed to tier because it was recalled at the same time.                                                                        | No action required. The file will be tiered when the recall completes and the file is no longer in use.                                                                                                                                                                                                           |
| 0x80c80264 | -2134375836       | ECS_E_GHOSTING_FILE_NOT_SYNCED | The file failed to tier because it has not synced to the Azure file share.                                                               | No action required. The file will tier once it has synced to the Azure file share.                                                                                                                                                                                                                                |

| HRESULT    | HRESULT (Decimal) | Error String                       | Issue                                                                                                      | Remediation                                                                                                                                                                                                                                                                                                                                         |
|------------|-------------------|------------------------------------|------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x80070001 | -2147942401       | ERROR_INVALID_FUNCTION             | The file failed to tier because the cloud tiering filter driver (storagesync.sys) is not running.          | <p>To resolve this issue, open an elevated command prompt and run the following command:</p> <pre data-bbox="1218 332 1350 384">fltmc load storagesync</pre> <p>If the storagesync filter driver fails to load when running the fltmc command, uninstall the Azure File Sync agent, restart the server and reinstall the Azure File Sync agent.</p> |
| 0x80070070 | -2147024784       | ERROR_DISK_FULL                    | The file failed to tier due to insufficient disk space on the volume where the server endpoint is located. | To resolve this issue, free at least 100 MB of disk space on the volume where the server endpoint is located.                                                                                                                                                                                                                                       |
| 0x80070490 | -2147023728       | ERROR_NOT_FOUND                    | The file failed to tier because it has not synced to the Azure file share.                                 | No action required. The file will tier once it has synced to the Azure file share.                                                                                                                                                                                                                                                                  |
| 0x80c80262 | -2134375838       | ECS_E_GHOSTING_UNSUPPORTED_RP      | The file failed to tier because it's an unsupported reparse point.                                         | If the file is a Data Deduplication reparse point, follow the steps in the <a href="#">planning guide</a> to enable Data Deduplication support. Files with reparse points other than Data Deduplication are not supported and will not be tiered.                                                                                                   |
| 0x80c83052 | -2134364078       | ECS_E_CREATE_SV_STREAM_ID_MISMATCH | The file failed to tier because it has been modified.                                                      | No action required. The file will tier once the modified file has synced to the Azure file share.                                                                                                                                                                                                                                                   |
| 0x80c80269 | -2134375831       | ECS_E_GHOSTING_REPLICA_NOT_FOUND   | The file failed to tier because it has not synced to the Azure file share.                                 | No action required. The file will tier once it has synced to the Azure file share.                                                                                                                                                                                                                                                                  |
| 0x80072ee2 | -2147012894       | WININET_E_TIMEOUT                  | The file failed to tier due to a network issue.                                                            | No action required. If the error persists, check network connectivity to the Azure file share.                                                                                                                                                                                                                                                      |

| HRESULT    | HRESULT (Decimal) | Error String              | Issue                                                         | Remediation                                                                                                |
|------------|-------------------|---------------------------|---------------------------------------------------------------|------------------------------------------------------------------------------------------------------------|
| 0x80c80017 | -2134376425       | ECS_E_SYNC_OPLOCK_BROKEN  | The file failed to tier because it has been modified.         | No action required. The file will tier once the modified file has synced to the Azure file share.          |
| 0x800705aa | -2147023446       | ERROR_NO_SYSTEM_RESOURCES | The file failed to tier due to insufficient system resources. | If the error persists, investigate which application or kernel-mode driver is exhausting system resources. |

### How to troubleshoot files that fail to be recalled

If files fail to be recalled:

1. In Event Viewer, review the telemetry, operational and diagnostic event logs, located under Applications and Services\Microsoft\FileSync\Agent.
  - a. Verify the files exist in the Azure file share.
  - b. Verify the server has internet connectivity.
  - c. Open the Services MMC snap-in and verify the Storage Sync Agent service (FileSyncSvc) is running.
  - d. Verify the Azure File Sync filter drivers (StorageSync.sys and StorageSyncGuard.sys) are running:
    - At an elevated command prompt, run `f1tmc`. Verify that the StorageSync.sys and StorageSyncGuard.sys file system filter drivers are listed.

#### NOTE

An Event ID 9006 is logged once per hour in the Telemetry event log if a file fails to recall (one event is logged per error code). Check the [Recall errors and remediation](#) section to see if remediation steps are listed for the error code.

### Recall errors and remediation

| HRESULT    | HRESULT (Decimal) | Error String       | Issue                                                                                                                                                                                                  | Remediation                                                                              |
|------------|-------------------|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------|
| 0x80070079 | -2147942521       | ERROR_SEM_TIMEOUT  | The file failed to recall due to an I/O timeout. This issue can occur for several reasons: server resource constraints, poor network connectivity or an Azure storage issue (for example, throttling). | No action required. If the error persists for several hours, please open a support case. |
| 0x80070036 | -2147024842       | ERROR_NETWORK_BUSY | The file failed to recall due to a network issue.                                                                                                                                                      | If the error persists, check network connectivity to the Azure file share.               |

| HRESULT    | HRESULT (Decimal) | Error String                                        | Issue                                                                                                                                                                                                                     | Remediation                                                                                                                                                                                                                                                        |
|------------|-------------------|-----------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x80c80037 | -2134376393       | ECS_E_SYNC_SHARE_NOT_FOUND                          | The file failed to recall because the server endpoint was deleted.                                                                                                                                                        | To resolve this issue, see <a href="#">Tiered files are not accessible on the server after deleting a server endpoint</a> .                                                                                                                                        |
| 0x80070005 | -2147024891       | ERROR_ACCESS_DENIED                                 | The file failed to recall due to an access denied error. This issue can occur if the firewall and virtual network settings on the storage account are enabled and the server does not have access to the storage account. | To resolve this issue, add the Server IP address or virtual network by following the steps documented in the <a href="#">Configure firewall and virtual network settings</a> section in the deployment guide.                                                      |
| 0x80c86002 | -2134351870       | ECS_E_AZURE_RESOURCE_NOT_FOUND                      | The file failed to recall because it's not accessible in the Azure file share.                                                                                                                                            | To resolve this issue, verify the file exists in the Azure file share. If the file exists in the Azure file share, upgrade to the latest Azure File Sync <a href="#">agent version</a> .                                                                           |
| 0x80c8305f | -2134364065       | ECS_E_EXTERNAL_STORAGE_ACCOUNT_AUTHORIZATION_FAILED | The file failed to recall due to authorization failure to the storage account.                                                                                                                                            | To resolve this issue, verify <a href="#">Azure File Sync has access to the storage account</a> .                                                                                                                                                                  |
| 0x80c86030 | -2134351824       | ECS_E_AZURE_FILE_SHARE_NOT_FOUND                    | The file failed to recall because the Azure file share is not accessible.                                                                                                                                                 | Verify the file share exists and is accessible. If the file share was deleted and recreated, perform the steps documented in the <a href="#">Sync failed because the Azure file share was deleted and recreated</a> section to delete and recreate the sync group. |
| 0x800705aa | -2147023446       | ERROR_NO_SYSTEM_RESOURCES                           | The file failed to recall due to insufficient system resources.                                                                                                                                                           | If the error persists, investigate which application or kernel-mode driver is exhausting system resources.                                                                                                                                                         |
| 0x8007000e | -2147024882       | ERROR_OUTOFMEMORY                                   | The file failed to recall due to insufficient memory.                                                                                                                                                                     | If the error persists, investigate which application or kernel-mode driver is causing the low memory condition.                                                                                                                                                    |

| HRESULT    | HRESULT (Decimal) | Error String    | Issue                                                     | Remediation                                                                                                                                                                                           |
|------------|-------------------|-----------------|-----------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0x80070070 | -2147024784       | ERROR_DISK_FULL | The file failed to recall due to insufficient disk space. | To resolve this issue, free up space on the volume by moving files to a different volume, increase the size of the volume, or force files to tier by using the Invoke-StorageSyncCloudTiering cmdlet. |

### Tiered files are not accessible on the server after deleting a server endpoint

Tiered files on a server will become inaccessible if the files are not recalled prior to deleting a server endpoint.

Errors logged if tiered files are not accessible

- When syncing a file, error code -2147942467 (0x80070043 - ERROR\_BAD\_NET\_NAME) is logged in the ItemResults event log
- When recalling a file, error code -2134376393 (0x80c80037 - ECS\_E\_SYNC\_SHARE\_NOT\_FOUND) is logged in the RecallResults event log

Restoring access to your tiered files is possible if the following conditions are met:

- Server endpoint was deleted within past 30 days
- Cloud endpoint was not deleted
- File share was not deleted
- Sync group was not deleted

If the above conditions are met, you can restore access to the files on the server by recreating the server endpoint at the same path on the server within the same sync group within 30 days.

If the above conditions are not met, restoring access is not possible as these tiered files on the server are now orphaned. Please follow the instructions below to remove the orphaned tiered files.

### Notes

- When tiered files are not accessible on the server, the full file should still be accessible if you access the Azure file share directly.
- To prevent orphaned tiered files in the future, follow the steps documented in [Remove a server endpoint](#) when deleting a server endpoint.

### How to get the list of orphaned tiered files

- Verify Azure File Sync agent version v5.1 or later is installed.
- Run the following PowerShell commands to list orphaned tiered files:

```
Import-Module "C:\Program Files\Azure\StorageSyncAgent\StorageSync.Management.ServerCmdlets.dll"
$orphanFiles = Get-StorageSyncOrphanedTieredFiles -path <server endpoint path>
$orphanFiles.OrphanedTieredFiles > OrphanTieredFiles.txt
```

- Save the OrphanTieredFiles.txt output file in case files need to be restored from backup after they are deleted.

### How to remove orphaned tiered files

*Option 1: Delete the orphaned tiered files*

This option deletes the orphaned tiered files on the Windows Server but requires removing the server endpoint if it exists due to recreation after 30 days or is connected to a different sync group. File conflicts will occur if files are updated on the Windows Server or Azure file share before the server endpoint is recreated.

1. Verify Azure File Sync agent version v5.1 or later is installed.
2. Backup the Azure file share and server endpoint location.
3. Remove the server endpoint in the sync group (if exists) by following the steps documented in [Remove a server endpoint](#).

#### WARNING

If the server endpoint is not removed prior to using the `Remove-StorageSyncOrphanedTieredFiles` cmdlet, deleting the orphaned tiered file on the server will delete the full file in the Azure file share.

4. Run the following PowerShell commands to list orphaned tiered files:

```
Import-Module "C:\Program Files\Azure\StorageSyncAgent\StorageSync.Management.ServerCmdlets.dll"
$orphanFiles = Get-StorageSyncOrphanedTieredFiles -path <server endpoint path>
$orphanFiles.OrphanedTieredFiles > OrphanTieredFiles.txt
```

5. Save the `OrphanTieredFiles.txt` output file in case files need to be restored from backup after they are deleted.
6. Run the following PowerShell commands to delete orphaned tiered files:

```
Import-Module "C:\Program Files\Azure\StorageSyncAgent\StorageSync.Management.ServerCmdlets.dll"
$orphanFilesRemoved = Remove-StorageSyncOrphanedTieredFiles -Path <folder path containing orphaned tiered files> -Verbose
$orphanFilesRemoved.OrphanedTieredFiles > DeletedOrphanFiles.txt
```

#### Notes

- Tiered files modified on the server that are not synced to the Azure file share will be deleted.
- Tiered files which are accessible (not orphan) will not be deleted.
- Non-tiered files will remain on the server.

7. Optional: Recreate the server endpoint if deleted in step 3.

#### *Option 2: Mount the Azure file share and copy the files locally that are orphaned on the server*

This option doesn't require removing the server endpoint but requires sufficient disk space to copy the full files locally.

1. [Mount](#) the Azure file share on the Windows Server that has orphaned tiered files.
2. Run the following PowerShell commands to list orphaned tiered files:

```
Import-Module "C:\Program Files\Azure\StorageSyncAgent\StorageSync.Management.ServerCmdlets.dll"
$orphanFiles = Get-StorageSyncOrphanedTieredFiles -path <server endpoint path>
$orphanFiles.OrphanedTieredFiles > OrphanTieredFiles.txt
```

3. Use the `OrphanTieredFiles.txt` output file to identify orphaned tiered files on the server.
4. Overwrite the orphaned tiered files by copying the full file from the Azure file share to the Windows Server.

#### **How to troubleshoot files unexpectedly recalled on a server**

Antivirus, backup, and other applications that read large numbers of files cause unintended recalls unless they respect the skip offline attribute and skip reading the content of those files. Skipping offline files for products that

support this option helps avoid unintended recalls during operations like antivirus scans or backup jobs.

Consult with your software vendor to learn how to configure their solution to skip reading offline files.

Unintended recalls also might occur in other scenarios, like when you are browsing files in File Explorer. Opening a folder that has cloud-tiered files in File Explorer on the server might result in unintended recalls. This is even more likely if an antivirus solution is enabled on the server.

**NOTE**

Use Event ID 9059 in the Telemetry event log to determine which application(s) is causing recalls. This event provides application recall distribution for a server endpoint and is logged once an hour.

## General troubleshooting

If you encounter issues with Azure File Sync on a server, start by completing the following steps:

1. In Event Viewer, review the telemetry, operational and diagnostic event logs.
  - Sync, tiering, and recall issues are logged in the telemetry, diagnostic and operational event logs under Applications and Services\Microsoft\FileSync\Agent.
  - Issues related to managing a server (for example, configuration settings) are logged in the operational and diagnostic event logs under Applications and Services\Microsoft\FileSync\Management.
2. Verify the Azure File Sync service is running on the server:
  - Open the Services MMC snap-in and verify that the Storage Sync Agent service (FileSyncSvc) is running.
3. Verify the Azure File Sync filter drivers (StorageSync.sys and StorageSyncGuard.sys) are running:
  - At an elevated command prompt, run `f1tmc`. Verify that the StorageSync.sys and StorageSyncGuard.sys file system filter drivers are listed.

If the issue is not resolved, run the AFSDiag tool:

1. Create a directory where the AFSDiag output will be saved (for example, C:\Output).

**NOTE**

AFSDiag will delete all content in the output directory prior to collecting logs. Specify an output location which does not contain data.

2. Open an elevated PowerShell window, and then run the following commands (press Enter after each command):

```
cd "c:\Program Files\Azure\StorageSyncAgent"
Import-Module .\afsdia.ps1
Debug-Afs c:\output # Note: Use the path created in step 1.
```

3. For the Azure File Sync kernel mode trace level, enter **1** (unless otherwise specified, to create more verbose traces), and then press Enter.
4. For the Azure File Sync user mode trace level, enter **1** (unless otherwise specified, to create more verbose traces), and then press Enter.
5. Reproduce the issue. When you're finished, enter **D**.
6. A .zip file that contains logs and trace files is saved to the output directory that you specified.

## See also

- [Monitor Azure File Sync](#)
- [Azure Files frequently asked questions](#)
- [Troubleshoot Azure Files problems in Windows](#)
- [Troubleshoot Azure Files problems in Linux](#)

# Azure file share – failed to delete files from Azure file share

11/19/2019 • 2 minutes to read • [Edit Online](#)

The failure to delete files from Azure File Share can have several symptoms:

## Symptom 1:

Failed to delete a file in azure file share due to one of the two issues below:

- The file marked for delete
- The specified resource may be in use by an SMB client

## Symptom 2:

Not enough quota is available to process this command

## Cause

Error 1816 occurs when you reach the upper limit of concurrent open handles allowed for a file, on the computer where the file share is being mounted. For more information, see the [Azure Storage performance and scalability checklist](#).

## Resolution

Reduce the number of concurrent open handles by closing some handles.

## Prerequisite

### Install the latest Azure PowerShell module

- [Install the Azure PowerShell module](#)

### Connect to Azure:

```
Connect-AzAccount
```

### Select the subscription of the target storage account:

```
Select-AzSubscription -subscriptionid "SubscriptionID"
```

### Create context for the target storage account:

```
$Context = New-AzStorageContext -StorageAccountName "StorageAccountName" -StorageAccountKey "StorageAccessKey"
```

### Get the current open handles of the file share:

```
Get-AzStorageFileHandle -Context $Context -ShareName "FileShareName" -Recursive
```

## Example result:

| HANDLEID         | PATH                | CLIENTIP          | CLIENTPORT | OPENTIME   | LASTRECONNECTTIME | FILEID                       | PARENTID                    | SESSIONID                   |
|------------------|---------------------|-------------------|------------|------------|-------------------|------------------------------|-----------------------------|-----------------------------|
| 2591012<br>29083 | ---                 | 10.222.10<br>.123 | 62758      | 2019-10-05 | 12:16:50Z         | 0                            | 0                           | 95077585<br>46259807<br>489 |
| 2591012<br>29131 | ---                 | 10.222.10<br>.123 | 62758      | 2019-10-05 | 12:36:20Z         | 0                            | 0                           | 95077585<br>46259807<br>489 |
| 2591012<br>29137 | ---                 | 10.222.10<br>.123 | 62758      | 2019-10-05 | 12:36:53Z         | 0                            | 0                           | 95077585<br>46259807<br>489 |
| 2591012<br>29136 | New folder/test.zip | 10.222.10<br>.123 | 62758      | 2019-10-05 | 12:36:29Z         | 13835132<br>82207285<br>2480 | 92234468<br>03645464<br>576 | 95077585<br>46259807<br>489 |
| 2591012<br>29135 | test.zip            | 37.222.22<br>.143 | 62758      | 2019-10-05 | 12:36:24Z         | 11529250<br>23044055<br>8592 | 0                           | 95077585<br>46259807<br>489 |

### Close an open handle:

To close an open handle, use the following command:

```
Close-AzStorageFileHandle -Context $Context -ShareName "FileShareName" -Path 'New folder/test.zip' -CloseAll
```

## Next steps

- [Troubleshoot Azure Files in Windows](#)
- [Troubleshoot Azure Files in Linux](#)
- [Troubleshoot Azure File Sync](#)

# How to recover a deleted storage account

11/20/2019 • 2 minutes to read • [Edit Online](#)

Azure Storage provides data resiliency through automated replicas, but doesn't prevent users or application code from corrupting data, whether accidentally or maliciously. Maintaining data fidelity during instances of application or user error requires more advanced techniques, such as copying the data to a secondary storage location with an audit log.

The following table provides overview of the scope of Storage Account Recovery depending on the replication strategy.

|                                              | LRS | ZRS | GRS | RA - GRS |
|----------------------------------------------|-----|-----|-----|----------|
| Storage Account<br>Azure Resource<br>Manager | Yes | Yes | Yes | Yes      |
| Storage Account<br>Classic                   | Yes | Yes | Yes | Yes      |

Gather the following information and file a support request with Microsoft Support:

- Storage account name
- Date of deletion
- Storage account region
- How was the storage account deleted?
- What method deleted the storage account? (Portal, PowerShell, etc.)

## Important Points

- It can generally take up to 15 days from the time of deletion for the storage service to perform garbage collection, so storage accounts recovery may not be recovered with an SLA.
- Microsoft Support will try to recover the Container/Account on a best-effort basis and cannot guarantee the recovery.

### NOTE

The recovery may not be successful if the account has been re-created. If you have already re-created the account, you must delete it first before recovery can be attempted.

# Release notes for the Azure File Sync agent

2/28/2020 • 39 minutes to read • [Edit Online](#)

Azure File Sync allows you to centralize your organization's file shares in Azure Files without giving up the flexibility, performance, and compatibility of an on-premises file server. Your Windows Server installations are transformed into a quick cache of your Azure file share. You can use any protocol that's available on Windows Server to access your data locally (including SMB, NFS, and FTPS). You can have as many caches as you need around the world.

This article provides the release notes for the supported versions of the Azure File Sync agent.

## Supported versions

The following versions are supported for the Azure File Sync agent:

| MILESTONE                                               | AGENT VERSION NUMBER | RELEASE DATE      | STATUS                                                  |
|---------------------------------------------------------|----------------------|-------------------|---------------------------------------------------------|
| December 2019 update rollup - <a href="#">KB4522360</a> | 9.1.0.0              | December 12, 2019 | Supported                                               |
| V9 Release - <a href="#">KB4522359</a>                  | 9.0.0.0              | December 2, 2019  | Supported                                               |
| V8 Release - <a href="#">KB4511224</a>                  | 8.0.0.0              | October 8, 2019   | Supported                                               |
| July 2019 update rollup - <a href="#">KB4490497</a>     | 7.2.0.0              | July 24, 2019     | Supported                                               |
| July 2019 update rollup - <a href="#">KB4490496</a>     | 7.1.0.0              | July 12, 2019     | Supported                                               |
| V7 Release - <a href="#">KB4490495</a>                  | 7.0.0.0              | June 19, 2019     | Supported                                               |
| June 2019 update rollup - <a href="#">KB4489739</a>     | 6.3.0.0              | June 27, 2019     | Supported                                               |
| June 2019 update rollup - <a href="#">KB4489738</a>     | 6.2.0.0              | June 13, 2019     | Supported                                               |
| May 2019 update rollup - <a href="#">KB4489737</a>      | 6.1.0.0              | May 7, 2019       | Supported                                               |
| V6 Release - <a href="#">KB4489736</a>                  | 6.0.0.0              | April 21, 2019    | Supported                                               |
| April 2019 update rollup - <a href="#">KB4481061</a>    | 5.2.0.0              | April 4, 2019     | Supported - Agent version will expire on March 18, 2020 |
| March 2019 update rollup - <a href="#">KB4481060</a>    | 5.1.0.0              | March 7, 2019     | Supported - Agent version will expire on March 18, 2020 |

| MILESTONE                              | AGENT VERSION NUMBER | RELEASE DATE      | STATUS                                                     |
|----------------------------------------|----------------------|-------------------|------------------------------------------------------------|
| V5 Release - <a href="#">KB4459989</a> | 5.0.2.0              | February 12, 2019 | Supported - Agent version will expire on March 18, 2020    |
| V4 Release                             | 4.0.1.0 - 4.3.0.0    | N/A               | Not Supported - Agent versions expired on November 6, 2019 |
| V3 Release                             | 3.1.0.0 - 3.4.0.0    | N/A               | Not Supported - Agent versions expired on August 19, 2019  |
| Pre-GA agents                          | 1.1.0.0 - 3.0.13.0   | N/A               | Not Supported - Agent versions expired on October 1, 2018  |

## Azure File Sync agent update policy

The Azure File Sync agent is updated on a regular basis to add new functionality and to address issues. We recommend you configure Microsoft Update to get updates for the Azure File Sync agent as they're available.

### Major vs. minor agent versions

- Major agent versions often contain new features and have an increasing number as the first part of the version number. For example: \*2.\*.\*
- Minor agent versions are also called "patches" and are released more frequently than major versions. They often contain bug fixes and smaller improvements but no new features. For example: \*\*.3.\*

### Upgrade paths

There are four approved and tested ways to install the Azure File Sync agent updates.

#### 1. (Preferred) Configure Microsoft Update to automatically download and install agent updates.

We always recommend taking every Azure File Sync update to ensure you have access to the latest fixes for the server agent. Microsoft Update makes this process seamless, by automatically downloading and installing updates for you.

#### 2. Use AfsUpdater.exe to download and install agent updates.

The AfsUpdater.exe is located in the agent installation directory. Double-click the executable to download and install agent updates.

#### 3. Patch an existing Azure File Sync agent by using a Microsoft Update patch file, or a .msp executable.

The latest Azure File Sync update package can be downloaded from the [Microsoft Update Catalog](#).

Running a .msp executable will upgrade your Azure File Sync installation with the same method used automatically by Microsoft Update in the previous upgrade path. Applying a Microsoft Update patch will perform an in-place upgrade of an Azure File Sync installation.

#### 4. Download the newest Azure File Sync agent installer from the [Microsoft Download Center](#).

To upgrade an existing Azure File Sync agent installation, uninstall the older version and then install the latest version from the downloaded installer. The server registration, sync groups, and any other settings are maintained by the Azure File Sync installer.

### Automatic agent lifecycle management

With agent version 6, the file sync team has introduced an agent auto-upgrade feature. You can select either of two modes and specify a maintenance window in which the upgrade shall be attempted on the server. This feature is designed to help you with the agent lifecycle management by either providing a guardrail preventing your agent from expiration or allowing for a no-hassle, stay current setting.

1. The **default setting** will attempt to prevent the agent from expiration. Within 21 days of the posted expiration

date of an agent, the agent will attempt to self-upgrade. It will start an attempt to upgrade once a week within 21 days prior to expiration and in the selected maintenance window. **This option does not eliminate the need for taking regular Microsoft Update patches.**

2. Optionally, you can select that the agent will automatically upgrade itself as soon as a new agent version becomes available (currently not applicable to clustered servers). This update will occur during the selected maintenance window and allow your server to benefit from new features and improvements as soon as they become generally available. This is the recommended, worry-free setting that will provide major agent versions as well as regular update patches to your server. Every agent released is at GA quality. If you select this option, Microsoft will flight the newest agent version to you. Clustered servers are excluded. Once flighting is complete, the agent will also become available on [Microsoft Download Center](#) aka.ms/AFS/agent.

#### Changing the auto-upgrade setting

The following instructions describe how to change the settings after you've completed the installer, if you need to make changes.

Open a PowerShell console and navigate to the directory where you installed the sync agent then import the server cmdlets. By default this would look something like this:

```
cd 'C:\Program Files\Azure\StorageSyncAgent'
Import-Module -Name .\StorageSync.Management.ServerCmdlets.dll
```

You can run `Get-StorageSyncAgentAutoUpdatePolicy` to check the current policy setting and determine if you want to change it.

To change the current policy setting to the delayed update track, you can use:

```
Set-StorageSyncAgentAutoUpdatePolicy -PolicyMode UpdateBeforeExpiration
```

To change the current policy setting to the immediate update track, you can use:

```
Set-StorageSyncAgentAutoUpdatePolicy -PolicyMode InstallLatest
```

#### Agent lifecycle and change management guarantees

Azure File Sync is a cloud service, which continuously introduces new features and improvements. This means that a specific Azure File Sync agent version can only be supported for a limited time. To facilitate your deployment, the following rules guarantee you have enough time and notification to accommodate agent updates/upgrades in your change management process:

- Major agent versions are supported for at least six months from the date of initial release.
- We guarantee there is an overlap of at least three months between the support of major agent versions.
- Warnings are issued for registered servers using a soon-to-be expired agent at least three months prior to expiration. You can check if a registered server is using an older version of the agent under the registered servers section of a Storage Sync Service.
- The lifetime of a minor agent version is bound to the associated major version. For example, when agent version 3.0 is released, agent versions 2.\* will all be set to expire together.

#### NOTE

Installing an agent version with an expiration warning will display a warning but succeed. Attempting to install or connect with an expired agent version is not supported and will be blocked.

## Agent version 9.1.0.0

The following release notes are for version 9.1.0.0 of the Azure File Sync agent released December 12, 2019. These notes are in addition to the release notes listed for version 9.0.0.0.

Issue fixed in this release:

- Sync fails with one of the following errors after upgrading to Azure File Sync agent version 9.0:
  - 0x8e5e044e (JET\_errWriteConflict)
  - 0x8e5e0450 (JET\_errInvalidSesid)
  - 0x8e5e0442 (JET\_errInstanceUnavailable)

## Agent version 9.0.0.0

The following release notes are for version 9.0.0.0 of the Azure File Sync agent (released December 2, 2019).

### Improvements and issues that are fixed

- Self-service restore support
  - Users can now restore their files by using the previous version feature. Prior to the v9 release, the previous version feature was not supported on volumes that had cloud tiering enabled. This feature must be enabled for each volume separately, on which an endpoint with cloud tiering enabled exists. To learn more, see [Self-service restore through Previous Versions and VSS \(Volume Shadow Copy Service\)](#).
- Support for larger file share sizes
  - Azure File Sync now supports up to 64TiB and 100 million files in a single, syncing namespace.
- Data Deduplication support on Server 2019
  - Data Deduplication is now supported with cloud tiering enabled on Windows Server 2019. To support Data Deduplication on volumes with cloud tiering, Windows update [KB4520062](#) must be installed.
- Improved minimum file size for a file to tier
  - The minimum file size for a file to tier is now based on the file system cluster size (double the file system cluster size). For example, by default, the NTFS file system cluster size is 4KB, the resulting minimum file size for a file to tier is 8KB.
- Network connectivity test cmdlet
  - As part of Azure File Sync configuration, multiple service endpoints must be contacted. They each have their own DNS name that needs to be accessible to the server. These URLs are also specific to the region a server is registered to. Once a server is registered, the connectivity test cmdlet (PowerShell and Server Registration Utility) can be used to test communications with all URLs specific to this server. This cmdlet can help troubleshoot when incomplete communication prevents the server from fully working with Azure File Sync and it can be used to fine tune proxy and firewall configurations.

To run the network connectivity test, run the following PowerShell commands:

```
Import-Module "C:\Program
Files\Azure\StorageSyncAgent\StorageSync.Management.ServerCmdlets.dll"
Test-StorageSyncNetworkConnectivity
```

- Remove server endpoint improvement when cloud tiering is enabled
  - As before, removing a server endpoint does not result in removing files in the Azure file share. However, behavior for reparse points on the local server has changed. Reparse points (pointers to files that are not local on the server) are now deleted when removing a server endpoint. The fully cached files will remain on the server. This improvement was made to prevent [orphaned tiered files](#) when removing a server

endpoint. If the server endpoint is recreated, the reparse points for the tiered files will be recreated on the server.

- Performance and reliability improvements
  - Reduced recall failures. Recall size is now automatically adjusted based on network bandwidth.
  - Improved download performance when adding a new server to a sync group.
  - Reduced files not syncing due to constraint conflicts.
  - Files fail to tier or are unexpectedly recalled in certain scenarios if the server endpoint path is a volume mount point.

## Evaluation Tool

Before deploying Azure File Sync, you should evaluate whether it is compatible with your system using the Azure File Sync evaluation tool. This tool is an Azure PowerShell cmdlet that checks for potential issues with your file system and dataset, such as unsupported characters or an unsupported OS version. For installation and usage instructions, see [Evaluation Tool](#) section in the planning guide.

## Agent installation and server configuration

For more information on how to install and configure the Azure File Sync agent with Windows Server, see [Planning for an Azure File Sync deployment](#) and [How to deploy Azure File Sync](#).

- The agent installation package must be installed with elevated (admin) permissions.
- The agent is not supported on Nano Server deployment option.
- The agent is supported only on Windows Server 2019, Windows Server 2016, and Windows Server 2012 R2.
- The agent requires at least 2 GiB of memory. If the server is running in a virtual machine with dynamic memory enabled, the VM should be configured with a minimum 2048 MiB of memory.
- The Storage Sync Agent (FileSyncSvc) service does not support server endpoints located on a volume that has the system volume information (SVI) directory compressed. This configuration will lead to unexpected results.

## Interoperability

- Antivirus, backup, and other applications that access tiered files can cause undesirable recall unless they respect the offline attribute and skip reading the content of those files. For more information, see [Troubleshoot Azure File Sync](#).
- File Server Resource Manager (FSRM) file screens can cause endless sync failures when files are blocked because of the file screen.
- Running sysprep on a server that has the Azure File Sync agent installed is not supported and can lead to unexpected results. The Azure File Sync agent should be installed after deploying the server image and completing sysprep mini-setup.

## Sync limitations

The following items don't sync, but the rest of the system continues to operate normally:

- Files with unsupported characters. See [Troubleshooting guide](#) for list of unsupported characters.
- Files or directories that end with a period.
- Paths that are longer than 2,048 characters.
- The discretionary access control list (DACL) portion of a security descriptor if it's larger than 2 KB. (This issue applies only when you have more than about 40 access control entries (ACEs) on a single item.)
- The system access control list (SACL) portion of a security descriptor that's used for auditing.
- Extended attributes.
- Alternate data streams.

- Reparse points.
- Hard links.
- Compression (if it's set on a server file) isn't preserved when changes sync to that file from other endpoints.
- Any file that's encrypted with EFS (or other user mode encryption) that prevents the service from reading the data.

**NOTE**

Azure File Sync always encrypts data in transit. Data is always encrypted at rest in Azure.

## Server endpoint

- A server endpoint can be created only on an NTFS volume. ReFS, FAT, FAT32, and other file systems aren't currently supported by Azure File Sync.
- Tiered files will become inaccessible if the files are not recalled prior to deleting the server endpoint. To restore access to the files, recreate the server endpoint. If 30 days have passed since the server endpoint was deleted or if the cloud endpoint was deleted, tiered files that were not recalled will be unusable. To learn more, see [Tiered files are not accessible on the server after deleting a server endpoint](#).
- Cloud tiering is not supported on the system volume. To create a server endpoint on the system volume, disable cloud tiering when creating the server endpoint.
- Failover Clustering is supported only with clustered disks, but not with Cluster Shared Volumes (CSVs).
- A server endpoint can't be nested. It can coexist on the same volume in parallel with another endpoint.
- Do not store an OS or application paging file within a server endpoint location.
- The server name in the portal is not updated if the server is renamed.

## Cloud endpoint

- Azure File Sync supports making changes to the Azure file share directly. However, any changes made on the Azure file share first need to be discovered by an Azure File Sync change detection job. A change detection job is initiated for a cloud endpoint once every 24 hours. To immediately sync files that are changed in the Azure file share, the [Invoke-AzStorageSyncChangeDetection](#) PowerShell cmdlet can be used to manually initiate the detection of changes in the Azure file share. In addition, changes made to an Azure file share over the REST protocol will not update the SMB last modified time and will not be seen as a change by sync.
- The storage sync service and/or storage account can be moved to a different resource group or subscription within the existing Azure AD tenant. If the storage account is moved, you need to give the Hybrid File Sync Service access to the storage account (see [Ensure Azure File Sync has access to the storage account](#)).

**NOTE**

Azure File Sync does not support moving the subscription to a different Azure AD tenant.

## Cloud tiering

- If a tiered file is copied to another location by using Robocopy, the resulting file isn't tiered. The offline attribute might be set because Robocopy incorrectly includes that attribute in copy operations.
- When copying files using robocopy, use the /MIR option to preserve file timestamps. This will ensure older files are tiered sooner than recently accessed files.
- Files may fail to tier if the pagefile.sys is located on a volume that has cloud tiering enabled. The pagefile.sys should be located on a volume that has cloud tiering disabled.

# Agent version 8.0.0.0

The following release notes are for version 8.0.0.0 of the Azure File Sync agent (released October 8, 2019).

## Improvements and issues that are fixed

- Restore performance Improvements
  - Faster recovery times for recovery done through Azure Backup. Restored files will sync back down to Azure File Sync servers much faster.
- Improved cloud tiering portal experience
  - If you have tiered files that are failing to recall, you can now view the recall errors in the server endpoint properties. Also, the server endpoint health will now show an error and mitigation steps if the cloud tiering filter driver is not loaded on the server.
- Simpler agent installation
  - The Az\AzureRM PowerShell module is no longer required to register the server making installation simpler and fast.
- Miscellaneous performance and reliability improvements

## Evaluation Tool

Before deploying Azure File Sync, you should evaluate whether it is compatible with your system using the Azure File Sync evaluation tool. This tool is an Azure PowerShell cmdlet that checks for potential issues with your file system and dataset, such as unsupported characters or an unsupported OS version. For installation and usage instructions, see [Evaluation Tool](#) section in the planning guide.

## Agent installation and server configuration

For more information on how to install and configure the Azure File Sync agent with Windows Server, see [Planning for an Azure File Sync deployment](#) and [How to deploy Azure File Sync](#).

- The agent installation package must be installed with elevated (admin) permissions.
- The agent is not supported on Nano Server deployment option.
- The agent is supported only on Windows Server 2019, Windows Server 2016, and Windows Server 2012 R2.
- The agent requires at least 2 GiB of memory. If the server is running in a virtual machine with dynamic memory enabled, the VM should be configured with a minimum 2048 MiB of memory.
- The Storage Sync Agent (FileSyncSvc) service does not support server endpoints located on a volume that has the system volume information (SVI) directory compressed. This configuration will lead to unexpected results.

## Interoperability

- Antivirus, backup, and other applications that access tiered files can cause undesirable recall unless they respect the offline attribute and skip reading the content of those files. For more information, see [Troubleshoot Azure File Sync](#).
- File Server Resource Manager (FSRM) file screens can cause endless sync failures when files are blocked because of the file screen.
- Running sysprep on a server that has the Azure File Sync agent installed is not supported and can lead to unexpected results. The Azure File Sync agent should be installed after deploying the server image and completing sysprep mini-setup.

## Sync limitations

The following items don't sync, but the rest of the system continues to operate normally:

- Files with unsupported characters. See [Troubleshooting guide](#) for list of unsupported characters.
- Files or directories that end with a period.
- Paths that are longer than 2,048 characters.

- The discretionary access control list (DACL) portion of a security descriptor if it's larger than 2 KB. (This issue applies only when you have more than about 40 access control entries (ACEs) on a single item.)
- The system access control list (SACL) portion of a security descriptor that's used for auditing.
- Extended attributes.
- Alternate data streams.
- Reparse points.
- Hard links.
- Compression (if it's set on a server file) isn't preserved when changes sync to that file from other endpoints.
- Any file that's encrypted with EFS (or other user mode encryption) that prevents the service from reading the data.

**NOTE**

Azure File Sync always encrypts data in transit. Data is always encrypted at rest in Azure.

## Server endpoint

- A server endpoint can be created only on an NTFS volume. ReFS, FAT, FAT32, and other file systems aren't currently supported by Azure File Sync.
- Tiered files will become inaccessible if the files are not recalled prior to deleting the server endpoint. To restore access to the files, recreate the server endpoint. If 30 days have passed since the server endpoint was deleted or if the cloud endpoint was deleted, tiered files that were not recalled will be unusable. To learn more, see [Tiered files are not accessible on the server after deleting a server endpoint](#).
- Cloud tiering is not supported on the system volume. To create a server endpoint on the system volume, disable cloud tiering when creating the server endpoint.
- Failover Clustering is supported only with clustered disks, but not with Cluster Shared Volumes (CSVs).
- A server endpoint can't be nested. It can coexist on the same volume in parallel with another endpoint.
- Do not store an OS or application paging file within a server endpoint location.
- The server name in the portal is not updated if the server is renamed.

## Cloud endpoint

- Azure File Sync supports making changes to the Azure file share directly. However, any changes made on the Azure file share first need to be discovered by an Azure File Sync change detection job. A change detection job is initiated for a cloud endpoint once every 24 hours. To immediately sync files that are changed in the Azure file share, the [Invoke-AzStorageSyncChangeDetection](#) PowerShell cmdlet can be used to manually initiate the detection of changes in the Azure file share. In addition, changes made to an Azure file share over the REST protocol will not update the SMB last modified time and will not be seen as a change by sync.
- The storage sync service and/or storage account can be moved to a different resource group or subscription within the existing Azure AD tenant. If the storage account is moved, you need to give the Hybrid File Sync Service access to the storage account (see [Ensure Azure File Sync has access to the storage account](#)).

**NOTE**

Azure File Sync does not support moving the subscription to a different Azure AD tenant.

## Cloud tiering

- If a tiered file is copied to another location by using Robocopy, the resulting file isn't tiered. The offline attribute might be set because Robocopy incorrectly includes that attribute in copy operations.
- When copying files using robocopy, use the /MIR option to preserve file timestamps. This will ensure older files are tiered sooner than recently accessed files.

## Agent version 7.2.0.0

The following release notes are for version 7.2.0.0 of the Azure File Sync agent released July 24, 2019. These notes are in addition to the release notes listed for version 7.0.0.0.

List of issues fixed in this release:

- Storage Sync Agent (FileSyncSvc) crashes if the proxy configuration is null.
- Server endpoint will start BCDR (error 0x80c80257 - ECS\_E\_BCDR\_IN\_PROGRESS) if multiple endpoints on the server have the same name.
- Cloud tiering reliability improvements.

## Agent version 7.1.0.0

The following release notes are for version 7.1.0.0 of the Azure File Sync agent released July 12, 2019. These notes are in addition to the release notes listed for version 7.0.0.0.

List of issues fixed in this release:

- Accessing or browsing a server endpoint location over SMB is slow on Windows Server 2012 R2.
- Increased CPU utilization after installing the Azure File Sync v6 agent.
- Cloud tiering telemetry improvements.
- Miscellaneous reliability improvements for cloud tiering and sync.

## Agent version 7.0.0.0

The following release notes are for version 7.0.0.0 of the Azure File Sync agent (released June 19, 2019).

### Improvements and issues that are fixed

- Support for larger file share sizes
  - With the preview of larger Azure file shares, we are increasing our support limits for file sync as well. In this first step, Azure File Sync now supports up to 25 TB and 50 million files in a single, syncing namespace. To apply for the large file share preview, fill in this form <https://aka.ms/azurefilesatscalesurvey>.
- Support for firewall and virtual network setting on storage accounts
  - Azure File Sync now supports the firewall and virtual network setting on storage accounts. To configure your deployment to work with the firewall and virtual network setting, see [Configure firewall and virtual network settings](#).
- PowerShell cmdlet to immediately sync files changed in the Azure file share
  - To immediately sync files that are changed in the Azure file share, the Invoke-AzStorageSyncChangeDetection PowerShell cmdlet can be used to manually initiate the detection of changes in the Azure file share. This cmdlet is intended for scenarios where some type of automated process is making changes in the Azure file share or the changes are done by an administrator (like moving files and directories into the share). For end-user changes, the recommendation is to install the Azure File Sync agent in an IaaS VM and have end users access the file share through the IaaS VM. This way all changes will quickly sync to other agents without the need to use the Invoke-AzStorageSyncChangeDetection cmdlet. To learn more, see the [Invoke-AzStorageSyncChangeDetection](#) documentation.

- Improved portal experience if you encounter files that are not syncing
  - If you have files that are failing to sync, we now differentiate between transient and persistent errors in the portal. Transient errors usually resolve themselves without the need for admin action. For example, a file that is currently in use will not sync until the file handle is closed. For persistent errors, we now show the number of files impacted by each error. The persistent error count is also displayed in the files not syncing column of all server endpoints in a sync group.
- Improved Azure Backup file-level restore
  - Individual files restored using Azure Backup are now detected and synced to the server endpoint faster.
- Improved cloud tiering recall cmdlet reliability
  - The Invoke-StorageSyncFileRecall cmdlet now allows customers to specify per file retry count and per file retry delay similar to robocopy. Previously, this cmdlet would recall all tiered files under a given path in random order. With the new -Order parameter, this cmdlet will recall the hottest data first and honor the cloud tiering policy (stop recalling if the date policy is met or the volume free space is met; whichever happens first).
- Support for TLS 1.2 only (TLS 1.0 and 1.1 is disabled)
  - Azure File Sync now supports using TLS 1.2 only on servers that have TLS 1.0 and 1.1 disabled. Prior to this improvement, server registration would fail if TLS 1.0 and 1.1 was disabled on the server.
- Miscellaneous performance and reliability improvements for sync and cloud tiering
  - There are several reliability and performance improvements in this release. Some of them are targeted to make cloud tiering more efficient and Azure File Sync as a whole work better in those situations when you have a bandwidth throttling schedule set.

## Evaluation Tool

Before deploying Azure File Sync, you should evaluate whether it is compatible with your system using the Azure File Sync evaluation tool. This tool is an Azure PowerShell cmdlet that checks for potential issues with your file system and dataset, such as unsupported characters or an unsupported OS version. For installation and usage instructions, see [Evaluation Tool](#) section in the planning guide.

## Agent installation and server configuration

For more information on how to install and configure the Azure File Sync agent with Windows Server, see [Planning for an Azure File Sync deployment](#) and [How to deploy Azure File Sync](#).

- The agent installation package must be installed with elevated (admin) permissions.
- The agent is not supported on Nano Server deployment option.
- The agent is supported only on Windows Server 2019, Windows Server 2016, and Windows Server 2012 R2.
- The agent requires at least 2 GiB of memory. If the server is running in a virtual machine with dynamic memory enabled, the VM should be configured with a minimum 2048 MiB of memory.
- The Storage Sync Agent (FileSyncSvc) service does not support server endpoints located on a volume that has the system volume information (SVI) directory compressed. This configuration will lead to unexpected results.

## Interoperability

- Antivirus, backup, and other applications that access tiered files can cause undesirable recall unless they respect the offline attribute and skip reading the content of those files. For more information, see [Troubleshoot Azure File Sync](#).
- File Server Resource Manager (FSRM) file screens can cause endless sync failures when files are blocked because of the file screen.
- Running sysprep on a server that has the Azure File Sync agent installed is not supported and can lead to unexpected results. The Azure File Sync agent should be installed after deploying the server image and completing sysprep mini-setup.

## Sync limitations

The following items don't sync, but the rest of the system continues to operate normally:

- Files with unsupported characters. See [Troubleshooting guide](#) for list of unsupported characters.
- Files or directories that end with a period.
- Paths that are longer than 2,048 characters.
- The discretionary access control list (DACL) portion of a security descriptor if it's larger than 2 KB. (This issue applies only when you have more than about 40 access control entries (ACEs) on a single item.)
- The system access control list (SACL) portion of a security descriptor that's used for auditing.
- Extended attributes.
- Alternate data streams.
- Reparse points.
- Hard links.
- Compression (if it's set on a server file) isn't preserved when changes sync to that file from other endpoints.
- Any file that's encrypted with EFS (or other user mode encryption) that prevents the service from reading the data.

**NOTE**

Azure File Sync always encrypts data in transit. Data is always encrypted at rest in Azure.

## Server endpoint

- A server endpoint can be created only on an NTFS volume. ReFS, FAT, FAT32, and other file systems aren't currently supported by Azure File Sync.
- Tiered files will become inaccessible if the files are not recalled prior to deleting the server endpoint. To restore access to the files, recreate the server endpoint. If 30 days have passed since the server endpoint was deleted or if the cloud endpoint was deleted, tiered files that were not recalled will be unusable.
- Cloud tiering is not supported on the system volume. To create a server endpoint on the system volume, disable cloud tiering when creating the server endpoint.
- Failover Clustering is supported only with clustered disks, but not with Cluster Shared Volumes (CSVs).
- A server endpoint can't be nested. It can coexist on the same volume in parallel with another endpoint.
- Do not store an OS or application paging file within a server endpoint location.
- The server name in the portal is not updated if the server is renamed.

## Cloud endpoint

- Azure File Sync supports making changes to the Azure file share directly. However, any changes made on the Azure file share first need to be discovered by an Azure File Sync change detection job. A change detection job is initiated for a cloud endpoint once every 24 hours. In addition, changes made to an Azure file share over the REST protocol will not update the SMB last modified time and will not be seen as a change by sync.
- The storage sync service and/or storage account can be moved to a different resource group or subscription within the existing Azure AD tenant. If the storage account is moved, you need to give the Hybrid File Sync Service access to the storage account (see [Ensure Azure File Sync has access to the storage account](#)).

**NOTE**

Azure File Sync does not support moving the subscription to a different Azure AD tenant.

## Cloud tiering

- If a tiered file is copied to another location by using Robocopy, the resulting file isn't tiered. The offline attribute might be set because Robocopy incorrectly includes that attribute in copy operations.
- When copying files using robocopy, use the /MIR option to preserve file timestamps. This will ensure older files are tiered sooner than recently accessed files.

## Agent version 6.3.0.0

The following release notes are for version 6.3.0.0 of the Azure File Sync agent released June 27, 2019. These notes are in addition to the release notes listed for version 6.0.0.0.

List of issues fixed in this release:

- Accessing or browsing a server endpoint location over SMB is slow on Windows Server 2012 R2
- Increased CPU utilization after installing the Azure File Sync v6 agent
- Cloud tiering telemetry improvements

## Agent version 6.2.0.0

The following release notes are for version 6.2.0.0 of the Azure File Sync agent released June 13, 2019. These notes are in addition to the release notes listed for version 6.0.0.0.

List of issues fixed in this release:

- After creating a server endpoint, High CPU usage may occur when background recall is downloading files to the server
- Sync and cloud tiering operations may fail with error ECS\_E\_SERVER\_CREDENTIAL\_NEEDED due to token expiration
- Recalling a file may fail if the URL to download the file contains reserved characters

## Agent version 6.1.0.0

The following release notes are for version 6.1.0.0 of the Azure File Sync agent released May 6, 2019. These notes are in addition to the release notes listed for version 6.0.0.0.

List of issues fixed in this release:

- Windows Admin Center fails to display the agent version and server endpoint configuration on servers which have Azure File Sync agent version 6.0 installed.

## Agent version 6.0.0.0

The following release notes are for version 6.0.0.0 of the Azure File Sync agent (released April 22, 2019).

### Improvements and issues that are fixed

- Agent auto-update support
  - We have heard your feedback and added an auto-update feature into the Azure File Sync server agent. For more information, see [Azure File Sync agent update policy](#).
- Support for Azure file share ACLs
  - Azure File Sync has always supported syncing ACLs between server endpoints but the ACLs were not synced to the cloud endpoint (Azure file share). This release adds support for syncing ACLs between server and cloud endpoints.
- Parallel upload and download sync sessions for a server endpoint
  - Server endpoints now support uploading and downloading files at the same time. No more waiting for a

download to complete so files can be uploaded to the Azure file share.

- New Cloud Tiering cmdlets to get volume and tiering status
  - Two new, server-local PowerShell cmdlets can now be used to obtain cloud tiering and file recall information. They make logging information from two event channels on the server available:
    - Get-StorageSyncFileTieringResult will list all files and their paths that haven't tiered and reports on the reason why.
    - Get-StorageSyncFileRecallResult reports all file recall events. It lists every file recalled and its path as well as success or error for that recall.
  - By default, both event channels can store up to 1 MB each – you can increase the amount of files reported by increasing the event channel size.
- Support for FIPS mode
  - Azure File Sync now supports enabling FIPS mode on servers which have the Azure File Sync agent installed.
    - Prior to enabling FIPS mode on your server, install the Azure File Sync agent and [PackageManagement module](#) on your server. If FIPS is already enabled on the server, [manually download the PackageManagement module](#) to your server.
- Miscellaneous reliability improvements for cloud tiering and sync

## Evaluation Tool

Before deploying Azure File Sync, you should evaluate whether it is compatible with your system using the Azure File Sync evaluation tool. This tool is an Azure PowerShell cmdlet that checks for potential issues with your file system and dataset, such as unsupported characters or an unsupported OS version. For installation and usage instructions, see [Evaluation Tool](#) section in the planning guide.

## Agent installation and server configuration

For more information on how to install and configure the Azure File Sync agent with Windows Server, see [Planning for an Azure File Sync deployment](#) and [How to deploy Azure File Sync](#).

- The agent installation package must be installed with elevated (admin) permissions.
- The agent is not supported on Nano Server deployment option.
- The agent is supported only on Windows Server 2019, Windows Server 2016, and Windows Server 2012 R2.
- The agent requires at least 2 GiB of memory. If the server is running in a virtual machine with dynamic memory enabled, the VM should be configured with a minimum 2048 MiB of memory.
- The Storage Sync Agent (FileSyncSvc) service does not support server endpoints located on a volume that has the system volume information (SVI) directory compressed. This configuration will lead to unexpected results.

## Interoperability

- Antivirus, backup, and other applications that access tiered files can cause undesirable recall unless they respect the offline attribute and skip reading the content of those files. For more information, see [Troubleshoot Azure File Sync](#).
- File Server Resource Manager (FSRM) file screens can cause endless sync failures when files are blocked because of the file screen.
- Running sysprep on a server which has the Azure File Sync agent installed is not supported and can lead to unexpected results. The Azure File Sync agent should be installed after deploying the server image and completing sysprep mini-setup.

## Sync limitations

The following items don't sync, but the rest of the system continues to operate normally:

- Files with unsupported characters. See [Troubleshooting guide](#) for list of unsupported characters.
- Files or directories that end with a period.

- Paths that are longer than 2,048 characters.
- The discretionary access control list (DACL) portion of a security descriptor if it's larger than 2 KB. (This issue applies only when you have more than about 40 access control entries (ACEs) on a single item.)
- The system access control list (SACL) portion of a security descriptor that's used for auditing.
- Extended attributes.
- Alternate data streams.
- Reparse points.
- Hard links.
- Compression (if it's set on a server file) isn't preserved when changes sync to that file from other endpoints.
- Any file that's encrypted with EFS (or other user mode encryption) that prevents the service from reading the data.

**NOTE**

Azure File Sync always encrypts data in transit. Data is always encrypted at rest in Azure.

## Server endpoint

- A server endpoint can be created only on an NTFS volume. ReFS, FAT, FAT32, and other file systems aren't currently supported by Azure File Sync.
- Tiered files will become inaccessible if the files are not recalled prior to deleting the server endpoint. To restore access to the files, recreate the server endpoint. If 30 days have passed since the server endpoint was deleted or if the cloud endpoint was deleted, tiered files that were not recalled will be unusable.
- Cloud tiering is not supported on the system volume. To create a server endpoint on the system volume, disable cloud tiering when creating the server endpoint.
- Failover Clustering is supported only with clustered disks, but not with Cluster Shared Volumes (CSVs).
- A server endpoint can't be nested. It can coexist on the same volume in parallel with another endpoint.
- Do not store an OS or application paging file within a server endpoint location.
- The server name in the portal is not updated if the server is renamed.

## Cloud endpoint

- Azure File Sync supports making changes to the Azure file share directly. However, any changes made on the Azure file share first need to be discovered by an Azure File Sync change detection job. A change detection job is initiated for a cloud endpoint once every 24 hours. In addition, changes made to an Azure file share over the REST protocol will not update the SMB last modified time and will not be seen as a change by sync.
- The storage sync service and/or storage account can be moved to a different resource group or subscription within the existing Azure AD tenant. If the storage account is moved, you need to give the Hybrid File Sync Service access to the storage account (see [Ensure Azure File Sync has access to the storage account](#)).

**NOTE**

Azure File Sync does not support moving the subscription to a different Azure AD tenant.

## Cloud tiering

- If a tiered file is copied to another location by using Robocopy, the resulting file isn't tiered. The offline attribute might be set because Robocopy incorrectly includes that attribute in copy operations.

- When copying files using robocopy, use the /MIR option to preserve file timestamps. This will ensure older files are tiered sooner than recently accessed files.
- When you're viewing file properties from an SMB client, the offline attribute might appear to be set incorrectly due to SMB caching of file metadata.

## Agent version 5.2.0.0

The following release notes are for version 5.2.0.0 of the Azure File Sync agent released April 4, 2019. These notes are in addition to the release notes listed for version 5.0.2.0.

List of issues fixed in this release:

- Reliability improvements for offline data transfer and data transfer resume features
- Sync telemetry improvements

## Agent version 5.1.0.0

The following release notes are for version 5.1.0.0 of the Azure File Sync agent released March 7, 2019. These notes are in addition to the release notes listed for version 5.0.2.0.

List of issues fixed in this release:

- Files may fail to sync with error 0x80c8031d (ECS\_E\_CONCURRENCY\_CHECK\_FAILED) if change enumeration is failing on the server
- If a sync session or file receives an error 0x80072f78 (WININET\_E\_INVALID\_SERVER\_RESPONSE), sync will now retry the operation
- Files may fail to sync with error 0x80c80203 (ECS\_E\_SYNC\_INVALID\_STAGED\_FILE)
- High memory usage may occur when recalling files
- Cloud tiering telemetry improvements

## Agent version 5.0.2.0

The following release notes are for version 5.0.2.0 of the Azure File Sync agent (released February 12, 2019).

### **Improvements and issues that are fixed**

- Support for Azure Government cloud
  - We have added preview support for the Azure Government cloud. This requires a white-listed subscription and a special agent download from Microsoft. To get access to the preview, please email us directly at [AzureFiles@microsoft.com](mailto:AzureFiles@microsoft.com).
- Support for Data Deduplication
  - Data Deduplication is now fully supported with cloud tiering enabled on Windows Server 2016 and Windows Server 2019. Enabling deduplication on a volume with cloud tiering enabled lets you cache more files on-premises without provisioning more storage.
- Support for offline data transfer (e.g. via Data Box)
  - Easily migrate large amounts of data into Azure File Sync via any means you choose. You can choose Azure Data Box, AzCopy and even third-party migration services. No need to use massive amounts of bandwidth to get your data into Azure, in the case of Data Box – simply mail it there! To learn more, see [Offline Data Transfer Docs](#).
- Improved sync performance
  - Customers with multiple server endpoints on the same volume may have experienced slow sync performance prior to this release. Azure File Sync creates a temporary VSS snapshot once a day on the server to sync files that have open handles. Sync now supports multiple server endpoints syncing on a volume when a VSS sync session is active. No more waiting for a VSS sync session to complete so sync

can resume on other server endpoints on the volume.

- Improved monitoring in the portal
  - Charts have been added in the Storage Sync Service portal to view:
    - Number of files synced
    - Size of data transferred
    - Number of files not syncing
    - Size of data recalled
    - Server connectivity status
  - To learn more, see [Monitor Azure File Sync](#).
- Improved scalability and reliability
  - Maximum number of file system objects (directories and files) in a directory has increased to 1,000,000. Previous limit was 200,000.
  - Sync will try to resume data transfer rather than retransmitting when a transfer is interrupted for large files

## Evaluation Tool

Before deploying Azure File Sync, you should evaluate whether it is compatible with your system using the Azure File Sync evaluation tool. This tool is an Azure PowerShell cmdlet that checks for potential issues with your file system and dataset, such as unsupported characters or an unsupported OS version. For installation and usage instructions, see [Evaluation Tool](#) section in the planning guide.

## Agent installation and server configuration

For more information on how to install and configure the Azure File Sync agent with Windows Server, see [Planning for an Azure File Sync deployment](#) and [How to deploy Azure File Sync](#).

- The agent installation package must be installed with elevated (admin) permissions.
- The agent is not supported on Windows Server Core or Nano Server deployment options.
- The agent is supported only on Windows Server 2019, Windows Server 2016, and Windows Server 2012 R2.
- The agent requires at least 2 GiB of memory. If the server is running in a virtual machine with dynamic memory enabled, the VM should be configured with a minimum 2048 MiB of memory.
- The Storage Sync Agent (FileSyncSvc) service does not support server endpoints located on a volume that has the system volume information (SVI) directory compressed. This configuration will lead to unexpected results.
- FIPS mode is not supported and must be disabled.

## Interoperability

- Antivirus, backup, and other applications that access tiered files can cause undesirable recall unless they respect the offline attribute and skip reading the content of those files. For more information, see [Troubleshoot Azure File Sync](#).
- File Server Resource Manager (FSRM) file screens can cause endless sync failures when files are blocked because of the file screen.
- Running sysprep on a server which has the Azure File Sync agent installed is not supported and can lead to unexpected results. The Azure File Sync agent should be installed after deploying the server image and completing sysprep mini-setup.

## Sync limitations

The following items don't sync, but the rest of the system continues to operate normally:

- Files with unsupported characters. See [Troubleshooting guide](#) for list of unsupported characters.
- Files or directories that end with a period.
- Paths that are longer than 2,048 characters.

- The discretionary access control list (DACL) portion of a security descriptor if it's larger than 2 KB. (This issue applies only when you have more than about 40 access control entries (ACEs) on a single item.)
- The system access control list (SACL) portion of a security descriptor that's used for auditing.
- Extended attributes.
- Alternate data streams.
- Reparse points.
- Hard links.
- Compression (if it's set on a server file) isn't preserved when changes sync to that file from other endpoints.
- Any file that's encrypted with EFS (or other user mode encryption) that prevents the service from reading the data.

**NOTE**

Azure File Sync always encrypts data in transit. Data is always encrypted at rest in Azure.

## Server endpoint

- A server endpoint can be created only on an NTFS volume. ReFS, FAT, FAT32, and other file systems aren't currently supported by Azure File Sync.
- Tiered files will become inaccessible if the files are not recalled prior to deleting the server endpoint. To restore access to the files, recreate the server endpoint. If 30 days have passed since the server endpoint was deleted or if the cloud endpoint was deleted, tiered files that were not recalled will be unusable.
- Cloud tiering is not supported on the system volume. To create a server endpoint on the system volume, disable cloud tiering when creating the server endpoint.
- Failover Clustering is supported only with clustered disks, but not with Cluster Shared Volumes (CSVs).
- A server endpoint can't be nested. It can coexist on the same volume in parallel with another endpoint.
- Do not store an OS or application paging file within a server endpoint location.
- The server name in the portal is not updated if the server is renamed.

## Cloud endpoint

- Azure File Sync supports making changes to the Azure file share directly. However, any changes made on the Azure file share first need to be discovered by an Azure File Sync change detection job. A change detection job is initiated for a cloud endpoint once every 24 hours. In addition, changes made to an Azure file share over the REST protocol will not update the SMB last modified time and will not be seen as a change by sync.
- The storage sync service and/or storage account can be moved to a different resource group or subscription within the existing Azure AD tenant. If the storage account is moved, you need to give the Hybrid File Sync Service access to the storage account (see [Ensure Azure File Sync has access to the storage account](#)).

**NOTE**

Azure File Sync does not support moving the subscription to a different Azure AD tenant.

## Cloud tiering

- If a tiered file is copied to another location by using Robocopy, the resulting file isn't tiered. The offline attribute might be set because Robocopy incorrectly includes that attribute in copy operations.
- When copying files using robocopy, use the /MIR option to preserve file timestamps. This will ensure older files

are tiered sooner than recently accessed files.

- When you're viewing file properties from an SMB client, the offline attribute might appear to be set incorrectly due to SMB caching of file metadata.