Azure File Share

Azure File Share is a cloud-based file sharing that enables high availability and scalability, file share can be accessed by a SMB/NFS or Azure File Share REST API. That means that multiple VMs can access the same files with both read and write access.

**Making it ideal for distributed applications and shared storage scenarios.**

**But a key difference between the traditional file share and Azure File Share is URL-based access using SAS tokens.**

**Key Points:**

* **Anywhere Access**: With the right permissions, files stored in Azure Files can be accessed **globally** using HTTPS.
* **SAS Tokens**:
  + Allow **fine-grained control** — you can specify:
    - **Which file(s)** can be accessed
    - **What operations** are allowed (read/write/delete/list)
    - **For how long** the access is valid
  + They are great for **temporary sharing** without exposing your storage keys.

File shares can be used for many common scenarios:

Lift-and-shift applications from on-premises to Azure cloud like when on-prem app using a file share.

Great question!

**✅ Real-Time Example Use Case for Azure File Share from On-prem to Azure:**

**Scenario: Shared Storage for Multiple VMs in a Lift-and-Shift Application**

**Use Case**:  
A company has a legacy .NET application that was previously hosted on-prem on multiple Windows servers. These servers relied on a shared network file share for storing configuration files, logs, and user-uploaded documents.

When migrating to Azure, instead of re-architecting the entire app, they lift and shift the VMs to Azure and use **Azure File Share** to replace the old on-prem SMB share.

**💡 How Azure File Share Helps:**

* **Mount the file share** to each Azure VM (Windows or Linux) using SMB protocol.
* Application continues to access and save data just like before — no code change needed.
* High availability and geo-redundancy options ensure the data is durable and accessible.
* Backups and snapshots can be taken easily from the Azure portal.

**🔧 Bonus:**

* They can **share a support log** file with a third-party vendor by generating a **read-only SAS URL** that expires in 24 hours.

Great follow-up! Let’s break those down with simple examples so it’s crystal clear.

**1. Configuration Files**

These are files that store settings your application needs to run.

**📝 Example:**

* **appsettings.json** or **web.config** in a .NET application.
* These may include:
  + Database connection strings
  + API keys
  + Feature flags (like turning logging on/off)

💡 **Why store them in Azure File Share?**  
 Multiple VMs or app instances can read the same config file, keeping things consistent.

**📄 2. Logs**

Log files track what your app is doing — super helpful for debugging and auditing.

**📝 Example:**

* **error.log**, **app-2025-04-20.log**
* These might Include:
  + User login activity
  + API errors (like "Database timeout")
  + Info about scheduled jobs

💡 **Why store them in Azure File Share?**  
Logs from multiple app servers can be written to the same location for central analysis.

**📁 3. User-Uploaded Documents**

Files uploaded by users via your app.

**📝 Example:**

* A user uploads a **PDF resume**, a **profile picture**, or a **scanned invoice**.
* Your app saves it to a shared file location so any backend service or admin portal can retrieve it.

💡 **Why store them in Azure File Share?**  
It acts like a cloud-based file server — easy to mount, scalable, and accessible across the app.

Let me know if you want to see how these might look inside a file structure or want to mock up a real example app using them!

🛠️ How to Mount a Single Azure File Share to Multiple Linux VMs

✅ Step 1: Create Azure File Share

Create a Storage Account:

A screenshot of a computer

AI-generated content may be incorrect.

Create a File Share:

A white box with black text

AI-generated content may be incorrect.

✅ Step 2: Get Storage Account Key

You’ll use this to authenticate your VMs.

A computer code on a white background

AI-generated content may be incorrect.

✅ Step 3: Mount File Share on Each Linux VM

Assuming the file share name **is myfileshare** and you have the storage key.

**Install CIFS Utils (on the Linux VM):**

A white sign with black text

AI-generated content may be incorrect.

**Create a mount point:**

A close-up of a card

AI-generated content may be incorrect.

**Mount the file share:**

A close up of a computer screen

AI-generated content may be incorrect.

✅ Repeat these mount steps on each VM, using the same storage account, file share, and mount path.

**\*\*Once mounted, your file share behaves like any local directory:\*\***

* /mnt/azurefileshare/logs/
* /mnt/azurefileshare/configs/

Now, **multiple Linux VMs** can read/write to it **simultaneously** — all changes are reflected instantly across them.

🧪 Test:

After mounting on all VMs:

* Upload a file from VM1 to /mnt/azurefileshare/test.txt
* Access it from VM2: cat /mnt/azurefileshare/test.txt ✅

🔐 Tip:

For better security, consider using Azure AD-based authentication or a private endpoint to limit access to only your VMs in a VNet.

Want a quick Bash script to automate the mount across multiple VMs or integrate it into your Terraform provisioning?

so, it’s perfect for **shared files** and **app state/configuration** that need to persist **outside of pods** but still be **mounted like a disk**.

Awesome! Here's a step-by-step YAML example to show how you can mount an **Azure File Share** into a pod running in **AKS**.

**🔹 Step 1: Create Azure File Share & Storage Account Key Secret**

A screenshot of a computer code

AI-generated content may be incorrect.

**🔹 Step 2: Define Persistent Volume (PV)**

**A screenshot of a computer

AI-generated content may be incorrect.**

**🔹 Step 3: Define Persistent Volume Claim (PVC)**

A screenshot of a computer program

AI-generated content may be incorrect.

**🔹 Step 4: Mount in a Pod**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Yes, exactly! /mnt/azure (or /mnt/azureshare if you name it that) is the mount path inside your container — essentially, the directory inside the container where the Azure File Share is attached. Which we created in the above configuration.**

**📌 Result:**

Your container can now read and write to /mnt/azure, and the data is stored in Azure File Share — shared across pods and persistent even if the pod restarts.

Let me know if you want to expand this for a multi-container app or as part of a Helm chart!

**You *should* use a Private Endpoint** to securely connect your AKS workloads to Azure File Storage, especially in **production environments**.

**By default, Azure File Share (backed by storage Account) is accessible over the public internet (even with SAS token). That’s not ideal for production environments.** A diagram of a computer network

AI-generated content may be incorrect.

**✅ How This Works:**

1. **Create a Private Endpoint** for your storage account (specifically for the file sub resource).
2. Azure assigns a **private IP** in your subnet (same or peered with AKS).
3. \*\*You also need a **Private DNS Zone** (privatelink.file.core.windows.net) and link it to your VNet.\*\*
4. Your AKS pods will resolve the storage account FQDN to the **private IP**, not public.

* Create a private endpoint

A screenshot of a computer program

AI-generated content may be incorrect.

**🧠 Notes:**

* If your AKS is in another subnet or VNet, ensure **VNet peering** is set up.
* If using **custom DNS**, forward lookups for \*.file.core.windows.net to Azure’s resolver.

Does AFS Optimized for Internal Use?

Yes.

**As a Cloud and DevOps Engineer,** my responsibilities around **Azure File Share** typically involve **configuring, securing, and maintaining** file shares to ensure high availability, performance, and compliance with organizational policies. Azure File Share is often leveraged for **internal application use cases**, such as:

* Hosting shared configuration files
* Application logs and metrics.
* Centralized storage for multiple VMs or containers
* Mounting as persistent volumes in AKS pods

While **external sharing** (e.g., with customers or third-party vendors) *is technically possible* using **Azure File Sync with on-prem servers**, **SAS tokens**, or **Private Endpoints + Azure AD/NTLM authentication**, it's **not ideal** due to:

* Complexity in secure access management
* Latency concerns over public networks
* Alternatives like **Azure Blob Storage (for external sharing)** being more suitable with native support for anonymous or token-based access and CDN integration

**TL;DR**

Yes — **Azure File Share is best suited for internal, controlled environments** (e.g., within applications or between internal services). External sharing is **possible but not ideal**, especially when compared to other services like Blob Storage which are better optimized for that.

Let me know if you want a table comparing Azure File Share vs Blob Storage for different use cases — might be handy for design docs or interviews.

**"** I mean **apps or users outside your control** — like a third-party vendor, public-facing clients, or systems not within your Azure tenant or network boundary.

**❌ When it's *not ideal*:**

* You want to serve files **directly to users over the internet**
* You need **CDN integration**, versioning, or automatic tiering
* You expect **high concurrent reads/writes** from unknown clients

For those cases, **Azure Blob Storage** is a better fit.