To connect to **Azure** using **PowerShell** with a **client certificate** (typically for service principal authentication), you can use the Connect-AzAccount cmdlet along with the certificate thumbprint.

Here’s the PowerShell code to do this securely:

### **✅ Pre-requisites**

1. A **service principal** created with a certificate (either self-signed or issued by a CA).
2. The certificate installed in the **CurrentUser\My** store.
3. The **thumbprint** of the certificate.
4. The **Application (client) ID**, **Tenant ID**, and **Subscription ID**.

### **🔐 PowerShell Code: Login with Client Certificate**

powershell

CopyEdit

# Variables  
$tenantId = "<Your-Tenant-ID>"  
$appId = "<Your-Application-Client-ID>"  
$certThumbprint = "<Your-Certificate-Thumbprint>"  
$subscriptionId = "<Your-Subscription-ID>"  
  
# Connect using certificate  
Connect-AzAccount `  
 -ServicePrincipal `  
 -Tenant $tenantId `  
 -ApplicationId $appId `  
 -CertificateThumbprint $certThumbprint `  
 -SubscriptionId $subscriptionId

### **📌 Notes**

* The certificate must be located in the **Current User's Personal store** (Cert:\CurrentUser\My).
* You can check for your certificate with:

powershell

CopyEdit

Get-ChildItem -Path Cert:\CurrentUser\My | Where-Object { $\_.Thumbprint -eq "<Your-Certificate-Thumbprint>" }

* If you are using a **PFX file**, you can import it first:

powershell

CopyEdit

$cert = Import-PfxCertificate -FilePath "C:\Path\To\Cert.pfx" -CertStoreLocation Cert:\CurrentUser\My

Let me know if you'd prefer to use the certificate from a file directly without importing it to the store, or if you're using a managed identity instead.

##### **You said:**

want to use the certificate from the file directly and it requires certificate password as well

###### **ChatGPT said:**

To authenticate to **Azure** using **PowerShell** with a **client certificate file (.pfx)** and a **password**, you can use the Connect-AzAccount cmdlet with the -Certificate parameter.

Here’s how you can do it:

### **✅ PowerShell Script: Connect Using Client Certificate File (.pfx)**

powershell

CopyEdit

# Variables  
$pfxPath = "C:\Path\To\YourCert.pfx"  
$certPassword = ConvertTo-SecureString -String "your-cert-password" -AsPlainText -Force  
$appId = "<Your-Application-Client-ID>"  
$tenantId = "<Your-Tenant-ID>"  
$subscriptionId = "<Your-Subscription-ID>"  
  
# Import the certificate from PFX file  
$cert = New-Object System.Security.Cryptography.X509Certificates.X509Certificate2  
$cert.Import($pfxPath, $certPassword, [System.Security.Cryptography.X509Certificates.X509KeyStorageFlags]::Exportable)  
  
# Connect to Azure using the certificate  
Connect-AzAccount `  
 -ServicePrincipal `  
 -ApplicationId $appId `  
 -TenantId $tenantId `  
 -Certificate $cert `  
 -SubscriptionId $subscriptionId

### **📌 Notes**

* Exportable flag is used so the private key can be used for authentication.
* This does **not** require you to import the certificate into the store.
* If you want to **secure the password** better, use Read-Host -AsSecureString or a Key Vault reference instead of hardcoding it.

Example for secure password prompt:

powershell

CopyEdit

$certPassword = Read-Host -Prompt "Enter certificate password" -AsSecureString

Let me know if you’d like to automate this further with Azure Key Vault to retrieve the certificate and password securely.

##### **You said:**

get me the whole code whole code using venafi certificate along certificate password

###### **ChatGPT said:**

If you're using a **Venafi-issued certificate** stored as a **.pfx file**, and want to authenticate to **Azure using PowerShell** with the **certificate file and password**, here's a complete script that securely handles the connection.

This script does **not** require the certificate to be imported into the Windows certificate store.

### **✅ Complete PowerShell Script: Connect to Azure using Venafi PFX Certificate**

powershell

CopyEdit

# ---------------------------------------------  
# Configuration: Set these variables  
# ---------------------------------------------  
$pfxPath = "C:\Path\To\VenafiIssuedCertificate.pfx" # Path to your Venafi-issued PFX file  
$appId = "<Your-Azure-AD-App-Client-ID>" # Application (client) ID of the service principal  
$tenantId = "<Your-Azure-AD-Tenant-ID>" # Azure AD Tenant ID  
$subscriptionId = "<Your-Azure-Subscription-ID>" # Azure Subscription ID  
  
# Prompt user for the certificate password securely  
$certPassword = Read-Host -Prompt "Enter certificate password" -AsSecureString  
  
# ---------------------------------------------  
# Load the certificate from file  
# ---------------------------------------------  
$cert = New-Object System.Security.Cryptography.X509Certificates.X509Certificate2  
$cert.Import(  
 $pfxPath,  
 $certPassword,  
 [System.Security.Cryptography.X509Certificates.X509KeyStorageFlags]::Exportable  
)  
  
# Optional: Print certificate info (for debug)  
Write-Host "Using Certificate Subject:" $cert.Subject  
Write-Host "Thumbprint:" $cert.Thumbprint  
  
# ---------------------------------------------  
# Connect to Azure with client certificate  
# ---------------------------------------------  
Connect-AzAccount `  
 -ServicePrincipal `  
 -ApplicationId $appId `  
 -TenantId $tenantId `  
 -Certificate $cert `  
 -SubscriptionId $subscriptionId  
  
# ---------------------------------------------  
# Validate Connection  
# ---------------------------------------------  
$context = Get-AzContext  
Write-Host "Connected to Azure subscription:" $context.Subscription.Name

### **🔒 Security Tips**

* Never hardcode the password in scripts.
* Use Read-Host -AsSecureString or pull it securely from **Azure Key Vault** if running in CI/CD or automation scenarios.
* Ensure the **service principal** in Azure AD has the correct **RBAC permissions** to access Azure resources.

Let me know if you want the same workflow using Azure Key Vault to retrieve the Venafi certificate and password at runtime (without storing on disk).

##### **You said:**

need the powershell code to upload folders/files to azure storage account using the same client certificate authentication. put all of these together in a single code file

###### **ChatGPT said:**

Here’s a **complete PowerShell script** that:

1. **Authenticates to Azure** using a **Venafi-issued .pfx certificate** (from file, with password).
2. **Connects to a specific Azure subscription.**
3. **Uploads folders/files** to an **Azure Blob Storage container** using Az.Storage module.

### **📄 Complete PowerShell Script: Authenticate with Certificate + Upload Files to Blob Storage**

powershell

CopyEdit

# ============================  
# Configuration Section  
# ============================  
  
# Certificate config  
$pfxPath = "C:\Path\To\VenafiIssuedCertificate.pfx" # Path to PFX certificate  
$appId = "<Your-Azure-AD-App-Client-ID>" # Azure AD App (client ID)  
$tenantId = "<Your-Azure-AD-Tenant-ID>" # Azure AD tenant  
$subscriptionId = "<Your-Azure-Subscription-ID>" # Azure subscription  
  
# Storage upload config  
$resourceGroupName = "<Your-Resource-Group-Name>"  
$storageAccountName = "<YourStorageAccountName>" # Must be globally unique  
$containerName = "<YourContainerName>" # Blob container name  
$localPathToUpload = "C:\Path\To\Local\FolderOrFile" # Folder or file to upload  
  
# ============================  
# Authentication  
# ============================  
  
# Prompt for certificate password securely  
$certPassword = Read-Host -Prompt "Enter certificate password" -AsSecureString  
  
# Load the certificate from the PFX file  
$cert = New-Object System.Security.Cryptography.X509Certificates.X509Certificate2  
$cert.Import($pfxPath, $certPassword, [System.Security.Cryptography.X509Certificates.X509KeyStorageFlags]::Exportable)  
  
# Login to Azure using the certificate  
Connect-AzAccount `  
 -ServicePrincipal `  
 -ApplicationId $appId `  
 -TenantId $tenantId `  
 -Certificate $cert `  
 -SubscriptionId $subscriptionId  
  
# ============================  
# Get Storage Context  
# ============================  
  
# Get the storage account key  
$storageAccountKey = (Get-AzStorageAccountKey -ResourceGroupName $resourceGroupName -Name $storageAccountName)[0].Value  
  
# Create the storage context  
$ctx = New-AzStorageContext -StorageAccountName $storageAccountName -StorageAccountKey $storageAccountKey  
  
# ============================  
# Upload File(s)  
# ============================  
  
# Check if it's a directory or file  
if (Test-Path $localPathToUpload -PathType Container) {  
 # Upload folder (recursively)  
 Get-ChildItem -Path $localPathToUpload -Recurse | ForEach-Object {  
 if (-not $\_.PSIsContainer) {  
 $relativePath = $\_.FullName.Substring($localPathToUpload.Length).TrimStart('\')  
 Set-AzStorageBlobContent -File $\_.FullName -Container $containerName -Blob $relativePath -Context $ctx | Out-Null  
 Write-Host "Uploaded file: $relativePath"  
 }  
 }  
}  
elseif (Test-Path $localPathToUpload -PathType Leaf) {  
 # Upload single file  
 $fileName = Split-Path -Path $localPathToUpload -Leaf  
 Set-AzStorageBlobContent -File $localPathToUpload -Container $containerName -Blob $fileName -Context $ctx | Out-Null  
 Write-Host "Uploaded file: $fileName"  
}  
else {  
 Write-Error "Path not found: $localPathToUpload"  
}  
  
# ============================  
# Done  
# ============================  
Write-Host "`n✅ Upload complete."

### **✅ Requirements**

* PowerShell modules:
  + Az.Accounts
  + Az.Storage
* Venafi certificate in .pfx format accessible on disk.
* Role assignment: The service principal must have **Storage Blob Data Contributor** on the storage account.

Let me know if you want to **upload to Azure Data Lake (ADLS Gen2)** or **read the certificate and password from Azure Key Vault** instead.

Here’s a **complete PowerShell script** that:

1. **Authenticates to Azure** using a **Venafi-issued .pfx certificate** (from file, with password).
2. **Connects to a specific Azure subscription.**
3. **Uploads folders/files** to an **Azure Blob Storage container** using Az.Storage module.

### **📄 Complete PowerShell Script: Authenticate with Certificate + Upload Files to Blob Storage**

powershell

CopyEdit

# ============================  
# Configuration Section  
# ============================  
  
# Certificate config  
$pfxPath = "C:\Path\To\VenafiIssuedCertificate.pfx" # Path to PFX certificate  
$appId = "<Your-Azure-AD-App-Client-ID>" # Azure AD App (client ID)  
$tenantId = "<Your-Azure-AD-Tenant-ID>" # Azure AD tenant  
$subscriptionId = "<Your-Azure-Subscription-ID>" # Azure subscription  
  
# Storage upload config  
$resourceGroupName = "<Your-Resource-Group-Name>"  
$storageAccountName = "<YourStorageAccountName>" # Must be globally unique  
$containerName = "<YourContainerName>" # Blob container name  
$localPathToUpload = "C:\Path\To\Local\FolderOrFile" # Folder or file to upload  
  
# ============================  
# Authentication  
# ============================  
  
# Prompt for certificate password securely  
$certPassword = Read-Host -Prompt "Enter certificate password" -AsSecureString  
  
# Load the certificate from the PFX file  
$cert = New-Object System.Security.Cryptography.X509Certificates.X509Certificate2  
$cert.Import($pfxPath, $certPassword, [System.Security.Cryptography.X509Certificates.X509KeyStorageFlags]::Exportable)  
  
# Login to Azure using the certificate  
Connect-AzAccount `  
 -ServicePrincipal `  
 -ApplicationId $appId `  
 -TenantId $tenantId `  
 -Certificate $cert `  
 -SubscriptionId $subscriptionId  
  
# ============================  
# Get Storage Context  
# ============================  
  
# Get the storage account key  
$storageAccountKey = (Get-AzStorageAccountKey -ResourceGroupName $resourceGroupName -Name $storageAccountName)[0].Value  
  
# Create the storage context  
$ctx = New-AzStorageContext -StorageAccountName $storageAccountName -StorageAccountKey $storageAccountKey  
  
# ============================  
# Upload File(s)  
# ============================  
  
# Check if it's a directory or file  
if (Test-Path $localPathToUpload -PathType Container) {  
 # Upload folder (recursively)  
 Get-ChildItem -Path $localPathToUpload -Recurse | ForEach-Object {  
 if (-not $\_.PSIsContainer) {  
 $relativePath = $\_.FullName.Substring($localPathToUpload.Length).TrimStart('\')  
 Set-AzStorageBlobContent -File $\_.FullName -Container $containerName -Blob $relativePath -Context $ctx | Out-Null  
 Write-Host "Uploaded file: $relativePath"  
 }  
 }  
}  
elseif (Test-Path $localPathToUpload -PathType Leaf) {  
 # Upload single file  
 $fileName = Split-Path -Path $localPathToUpload -Leaf  
 Set-AzStorageBlobContent -File $localPathToUpload -Container $containerName -Blob $fileName -Context $ctx | Out-Null  
 Write-Host "Uploaded file: $fileName"  
}  
else {  
 Write-Error "Path not found: $localPathToUpload"  
}  
  
# ============================  
# Done  
# ============================  
Write-Host "`n✅ Upload complete."

### **✅ Requirements**

* PowerShell modules:
  + Az.Accounts
  + Az.Storage
* Venafi certificate in .pfx format accessible on disk.
* Role assignment: The service principal must have **Storage Blob Data Contributor** on the storage account.

Let me know if you want to **upload to Azure Data Lake (ADLS Gen2)** or **read the certificate and password from Azure Key Vault** instead.

##### **You said:**

get me the download powershell code from an azure storage account container/folder/file to a local path

###### **ChatGPT said:**

Here's the **PowerShell code** to **download files or folders** from an **Azure Storage Account blob container** (including support for folders within containers) to a **local path**.

This version assumes you're **authenticated using a service principal with client certificate** (as in your previous code), and then uses that context to download blobs.

### **📄 PowerShell Script: Download from Azure Blob Storage to Local Path**

powershell

CopyEdit

# ============================  
# Configuration Section  
# ============================  
  
# Certificate auth config  
$pfxPath = "C:\Path\To\VenafiIssuedCertificate.pfx" # Path to Venafi-issued PFX certificate  
$appId = "<Your-Azure-AD-App-Client-ID>" # Azure AD App (client ID)  
$tenantId = "<Your-Azure-AD-Tenant-ID>" # Azure AD tenant  
$subscriptionId = "<Your-Azure-Subscription-ID>" # Azure subscription  
  
# Storage account config  
$resourceGroupName = "<Your-Resource-Group-Name>"  
$storageAccountName = "<YourStorageAccountName>" # Storage account name  
$containerName = "<YourContainerName>" # Blob container name  
  
# Target for download  
$remotePrefix = "<optional/folder/prefix/in/container>" # Prefix inside container (e.g., "folder1/") or "" for all  
$localDownloadPath = "C:\Path\To\Local\DownloadFolder" # Local path to save blobs  
  
# ============================  
# Authentication  
# ============================  
  
# Prompt for certificate password securely  
$certPassword = Read-Host -Prompt "Enter certificate password" -AsSecureString  
  
# Load certificate from PFX  
$cert = New-Object System.Security.Cryptography.X509Certificates.X509Certificate2  
$cert.Import($pfxPath, $certPassword, [System.Security.Cryptography.X509Certificates.X509KeyStorageFlags]::Exportable)  
  
# Connect to Azure  
Connect-AzAccount `  
 -ServicePrincipal `  
 -ApplicationId $appId `  
 -TenantId $tenantId `  
 -Certificate $cert `  
 -SubscriptionId $subscriptionId  
  
# ============================  
# Get Storage Context  
# ============================  
  
# Get the storage account key  
$storageAccountKey = (Get-AzStorageAccountKey -ResourceGroupName $resourceGroupName -Name $storageAccountName)[0].Value  
  
# Create the storage context  
$ctx = New-AzStorageContext -StorageAccountName $storageAccountName -StorageAccountKey $storageAccountKey  
  
# ============================  
# Download Blobs  
# ============================  
  
# List blobs with optional prefix  
$blobs = Get-AzStorageBlob -Container $containerName -Context $ctx -Prefix $remotePrefix  
  
foreach ($blob in $blobs) {  
 $relativePath = $blob.Name  
 $localFilePath = Join-Path $localDownloadPath $relativePath  
  
 # Ensure local directory exists  
 $localDir = Split-Path -Path $localFilePath -Parent  
 if (-not (Test-Path -Path $localDir)) {  
 New-Item -Path $localDir -ItemType Directory -Force | Out-Null  
 }  
  
 # Download the blob  
 Get-AzStorageBlobContent `  
 -Blob $blob.Name `  
 -Container $containerName `  
 -Destination $localFilePath `  
 -Context $ctx `  
 -Force | Out-Null  
  
 Write-Host "Downloaded: $relativePath"  
}  
  
# ============================  
# Done  
# ============================  
Write-Host "`n✅ Download complete."

### **🔧 Customizations**

* Set $remotePrefix = "" to download the entire container.
* Handles **folders** by preserving blob virtual folder structure.
* You can modify it to **download only specific extensions** (e.g., .csv, .json) by filtering $blobs.

Let me know if you want to enhance this to:

* Only download blobs updated after a certain date.
* Handle Azure Data Lake Gen2.
* Use a **managed identity** or **Azure CLI login** instead.

##### **You said:**

yes, enhance to handle Azure data lake gen2

###### **ChatGPT said:**

To enhance the script for **Azure Data Lake Storage Gen2 (ADLS Gen2)**, we must:

* Use **Az.DataLakeStore** and **Az.Storage** modules.
* ADLS Gen2 uses **Hierarchical Namespace (HNS)** which means directories and files are treated as first-class objects.
* Use **Get-AzDataLakeGen2Item** and **Get-AzDataLakeGen2ChildItem** to list files.
* Use **Export-AzDataLakeGen2Item** to download files.

Authentication with client certificate stays the same, using Connect-AzAccount with -Certificate.

### **✅ Enhanced PowerShell Script: Download from Azure Data Lake Gen2 using Client Certificate**

powershell

CopyEdit

# ============================  
# Configuration Section  
# ============================  
  
# Certificate auth config  
$pfxPath = "C:\Path\To\VenafiIssuedCertificate.pfx"  
$appId = "<Your-Azure-AD-App-Client-ID>"  
$tenantId = "<Your-Azure-AD-Tenant-ID>"  
$subscriptionId = "<Your-Azure-Subscription-ID>"  
  
# ADLS Gen2 config  
$resourceGroupName = "<Your-Resource-Group-Name>"  
$storageAccountName = "<YourStorageAccountName>" # ADLS Gen2 account name  
$filesystemName = "<YourFileSystemName>" # ADLS Gen2 container (aka file system)  
$remoteDirectoryPath = "<optional/path/in/adls>" # e.g., "raw/2024/" or "" for root  
$localDownloadPath = "C:\Path\To\Local\DownloadFolder"  
  
# ============================  
# Authentication  
# ============================  
  
$certPassword = Read-Host -Prompt "Enter certificate password" -AsSecureString  
  
$cert = New-Object System.Security.Cryptography.X509Certificates.X509Certificate2  
$cert.Import($pfxPath, $certPassword, [System.Security.Cryptography.X509Certificates.X509KeyStorageFlags]::Exportable)  
  
Connect-AzAccount `  
 -ServicePrincipal `  
 -ApplicationId $appId `  
 -TenantId $tenantId `  
 -Certificate $cert `  
 -SubscriptionId $subscriptionId  
  
# ============================  
# Download Files Recursively from ADLS Gen2  
# ============================  
  
function Download-ADLSGen2Folder {  
 param (  
 [string]$storageAccountName,  
 [string]$filesystemName,  
 [string]$path,  
 [string]$localPath  
 )  
  
 $items = Get-AzDataLakeGen2ChildItem `  
 -AccountName $storageAccountName `  
 -FileSystem $filesystemName `  
 -Path $path `  
 -ErrorAction Stop  
  
 foreach ($item in $items) {  
 $remotePath = if ($path -ne "") { "$path/$($item.Name)" } else { $item.Name }  
 $localItemPath = Join-Path $localPath $item.Name  
  
 if ($item.IsDirectory) {  
 # Create local folder  
 if (-not (Test-Path $localItemPath)) {  
 New-Item -Path $localItemPath -ItemType Directory | Out-Null  
 }  
  
 # Recurse  
 Download-ADLSGen2Folder -storageAccountName $storageAccountName `  
 -filesystemName $filesystemName `  
 -path $remotePath `  
 -localPath $localItemPath  
 } else {  
 # Download file  
 Export-AzDataLakeGen2Item `  
 -AccountName $storageAccountName `  
 -FileSystem $filesystemName `  
 -Path $remotePath `  
 -Destination $localItemPath `  
 -Force  
  
 Write-Host "Downloaded: $remotePath"  
 }  
 }  
}  
  
# Create local base folder if needed  
if (-not (Test-Path $localDownloadPath)) {  
 New-Item -Path $localDownloadPath -ItemType Directory | Out-Null  
}  
  
# Start download  
Download-ADLSGen2Folder -storageAccountName $storageAccountName `  
 -filesystemName $filesystemName `  
 -path $remoteDirectoryPath `  
 -localPath $localDownloadPath  
  
Write-Host "`n✅ ADLS Gen2 download complete."

### **🔧 Requirements**

* Az.Accounts, Az.Storage, and Az.DataLakeStore modules.
* The ADLS Gen2 file system is usually the **container** name when hierarchical namespace is enabled.
* The service principal must have **Storage Blob Data Reader** or **Storage Data Reader** role at the container or file system level.

Let me know if you also want:

* Upload functionality for ADLS Gen2.
* Filtering by file type or last modified date.
* Integration with Azure Key Vault to fetch the certificate and password.

##### **You said:**

yes, require upload functionality for ADLS Gen2

###### **ChatGPT said:**

Here is the **PowerShell script** to **upload files/folders** to **Azure Data Lake Storage Gen2 (ADLS Gen2)** using **client certificate-based authentication** (e.g., Venafi .pfx), completing your round-trip workflow (auth ➜ upload ➜ download).

### **📦 PowerShell Script: Upload to Azure Data Lake Gen2 using Client Certificate**

powershell

CopyEdit

# ============================  
# Configuration Section  
# ============================  
  
# Certificate auth config  
$pfxPath = "C:\Path\To\VenafiIssuedCertificate.pfx"  
$appId = "<Your-Azure-AD-App-Client-ID>"  
$tenantId = "<Your-Azure-AD-Tenant-ID>"  
$subscriptionId = "<Your-Azure-Subscription-ID>"  
  
# ADLS Gen2 config  
$resourceGroupName = "<Your-Resource-Group-Name>"  
$storageAccountName = "<YourStorageAccountName>" # ADLS Gen2 account name  
$filesystemName = "<YourFileSystemName>" # ADLS Gen2 container (file system)  
$remoteTargetPath = "<optional/remote/folder>" # Path within ADLS (e.g., "raw/upload1")  
$localUploadPath = "C:\Path\To\Local\FolderOrFile" # Local file or folder to upload  
  
# ============================  
# Authentication  
# ============================  
  
$certPassword = Read-Host -Prompt "Enter certificate password" -AsSecureString  
  
$cert = New-Object System.Security.Cryptography.X509Certificates.X509Certificate2  
$cert.Import($pfxPath, $certPassword, [System.Security.Cryptography.X509Certificates.X509KeyStorageFlags]::Exportable)  
  
Connect-AzAccount `  
 -ServicePrincipal `  
 -ApplicationId $appId `  
 -TenantId $tenantId `  
 -Certificate $cert `  
 -SubscriptionId $subscriptionId  
  
# ============================  
# Upload File(s)/Folder to ADLS Gen2  
# ============================  
  
function Upload-ADLSGen2Folder {  
 param (  
 [string]$localBasePath,  
 [string]$remoteBasePath,  
 [string]$storageAccountName,  
 [string]$filesystemName  
 )  
  
 $items = Get-ChildItem -Path $localBasePath -Recurse -File  
  
 foreach ($item in $items) {  
 # Calculate relative path  
 $relativePath = $item.FullName.Substring($localBasePath.Length).TrimStart('\')  
 $remotePath = if ($remoteBasePath -ne "") { "$remoteBasePath/$relativePath" } else { $relativePath }  
 $remotePath = $remotePath -replace '\\','/' # Ensure Unix-style path  
  
 # Upload file  
 Import-AzDataLakeGen2Item `  
 -FileSystem $filesystemName `  
 -Path $remotePath `  
 -Source $item.FullName `  
 -AccountName $storageAccountName `  
 -Force  
  
 Write-Host "Uploaded: $relativePath --> $remotePath"  
 }  
}  
  
# If path is a folder, recurse and upload  
if (Test-Path $localUploadPath -PathType Container) {  
 Upload-ADLSGen2Folder `  
 -localBasePath $localUploadPath `  
 -remoteBasePath $remoteTargetPath `  
 -storageAccountName $storageAccountName `  
 -filesystemName $filesystemName  
}  
elseif (Test-Path $localUploadPath -PathType Leaf) {  
 $fileName = Split-Path -Path $localUploadPath -Leaf  
 $remoteBlobPath = if ($remoteTargetPath -ne "") { "$remoteTargetPath/$fileName" } else { $fileName }  
 $remoteBlobPath = $remoteBlobPath -replace '\\','/'  
  
 Import-AzDataLakeGen2Item `  
 -FileSystem $filesystemName `  
 -Path $remoteBlobPath `  
 -Source $localUploadPath `  
 -AccountName $storageAccountName `  
 -Force  
  
 Write-Host "Uploaded file: $remoteBlobPath"  
}  
else {  
 Write-Error "Path not found: $localUploadPath"  
}  
  
Write-Host "`n✅ ADLS Gen2 upload complete."

### **✅ Capabilities**

* Handles both single file and recursive folder upload.
* Preserves folder structure within the remote filesystem.
* Uses service principal with **client certificate (.pfx)**.
* Works for ADLS Gen2 with **Hierarchical Namespace enabled**.

### **🔐 Required Role**

Ensure the service principal has:

* **Storage Blob Data Contributor** or **Storage Data Contributor** on the container or storage account level.

Let me know if you'd like to:

* Include **Azure Key Vault integration** for secure certificate retrieval.
* Add **file type filters** or **parallel uploads** for large datasets.
* Package this as a **reusable PowerShell function module**