Here are various ways to move data between on-premises environments and Azure services like ADLS, Databricks, and Purview:

### **1. Moving Data from On-Premises to Azure**

#### **A. Using Azure Data Factory (ADF) / Synapse Pipelines**

* **Copy Activity**: Moves data from on-prem sources (SQL Server, Oracle, file shares) to ADLS, Blob Storage, or Databricks.
* **Self-hosted Integration Runtime (SHIR)**: Required for secure connectivity to on-prem sources.
* **Mapping Data Flows**: Transform data before loading it into Azure.

#### **B. Using Azure Data Box**

* **Azure Data Box / Data Box Disk / Data Box Heavy**: Physical devices for bulk data migration when network bandwidth is limited.

#### **C. Using Azure Storage Services**

* **AzCopy**: CLI tool to copy data from on-prem to ADLS or Blob Storage.
* **Azure Storage Explorer**: GUI-based tool for uploading files to Azure Storage.
* **SMB/NFS Mount**: Connect ADLS Gen2 or Blob Storage as a mounted drive.

#### **D. Using Azure Databricks**

* **Databricks Auto Loader**: Streams data from ADLS, Blob Storage, or Event Hubs.
* **Mounting ADLS**: Mount ADLS Gen2 or Blob Storage in Databricks.
* **Delta Live Tables (DLT)**: Continuous ETL for streaming/batch data.

#### **E. Using Event-Driven Services**

* **Azure Event Hubs**: Stream real-time data into Azure from on-prem applications.
* **Azure IoT Hub**: For IoT devices sending data to Azure.
* **Kafka on Azure**: Stream data from on-prem Kafka clusters to Azure.

#### **F. Direct Database Replication**

* **Azure Database Migration Service**: Move on-prem databases to Azure SQL, Synapse, or Cosmos DB.
* **Transactional Replication**: Sync SQL Server data with Azure SQL DB.
* **Change Data Capture (CDC) / Change Tracking**: Capture changes for incremental migration.

#### **G. Using Azure File Sync**

* **Sync On-Prem File Servers with Azure**: Extends on-prem file shares to Azure.

#### **H. Using Open-Source Tools**

* **rsync / SCP**: Move files from on-prem to Azure VMs.
* **Apache NiFi**: Automates data movement workflows.

### **2. Moving Data from Azure to On-Premises**

* **Azure Data Factory (ADF) with SHIR**: Extract data from Azure and write to on-prem sources.
* **AzCopy / Storage Explorer**: Download data from ADLS/Blob to on-prem storage.
* **Event Hubs / Kafka**: Stream data from Azure to on-prem systems.
* **Database Replication**: Use SQL Server transactional replication or CDC.
* **Azure File Sync**: Keep Azure and on-prem file shares synchronized.

Here’s a detailed breakdown of **AzCopy, WanDisco, SFTP, and DistCp** for moving data from on-prem to Azure, including their **REST API calls, use cases, and performance comparisons**:

## **1. AzCopy**

### **Overview**

AzCopy is a command-line utility designed for **high-speed, efficient data transfer** between on-premises storage and Azure Blob Storage or ADLS Gen2.

### **Use Cases**

* Bulk data movement from on-prem to Azure Blob Storage or ADLS Gen2.
* Incremental data transfer with **sync** functionality.
* Migration of large datasets using parallelized transfer.

### **Key REST API Calls**

AzCopy interacts with Azure Storage via REST APIs:

1. **Upload File to Blob Storage**

http

CopyEdit

PUT https://<storageaccount>.blob.core.windows.net/<container>/<blob>?sv=<SAS\_token>

1. **List Blobs in a Container**

http

CopyEdit

GET https://<storageaccount>.blob.core.windows.net/<container>?restype=container&comp=list

1. **Delete Blob**

http

CopyEdit

DELETE https://<storageaccount>.blob.core.windows.net/<container>/<blob>

### **Performance**

* **Optimized for parallel and multi-threaded uploads**.
* Can handle large files and **terabyte-scale transfers** efficiently.
* Supports **resume on failure** and **automatic retry mechanisms**.
* Uses **block-based transfer** for better speed.

## **2. WanDisco LiveData Migrator**

### **Overview**

WanDisco LiveData Migrator is an enterprise-grade solution for **real-time, continuous data replication** across hybrid cloud environments.

### **Use Cases**

* **Streaming, real-time migration** of HDFS, object storage, and other large datasets.
* **Zero downtime migration** with active-active replication.
* Synchronizing on-prem HDFS clusters with **Azure Data Lake Storage Gen2**.

### **Key REST API Calls**

WanDisco provides a REST API for automation:

1. **Create a Migration Policy**

http

CopyEdit

POST /api/migration  
{  
 "source": "hdfs://on-prem-cluster",  
 "target": "[abfs://[container]@[storageaccount].dfs.core.windows.net](mailto:abfs://[container]@[storageaccount].dfs.core.windows.net)",  
 "policy": "continuous"  
}

1. **Check Migration Status**

http

CopyEdit

GET /api/migration/status

1. **Cancel a Migration**

http

CopyEdit

DELETE /api/migration/<migration\_id>

### **Performance**

* **High-speed, real-time migration** with **no service disruption**.
* **No need for batch jobs**—keeps data **continuously synchronized**.
* Uses **smart compression and deduplication** for bandwidth efficiency.

## **3. Secure File Transfer Protocol (SFTP)**

### **Overview**

SFTP is a secure protocol used to **transfer files over SSH**. Azure provides **SFTP support on ADLS Gen2** for seamless data movement.

### **Use Cases**

* **Moving structured and unstructured files** from on-prem to Azure Storage.
* **Integration with legacy applications** that support SFTP.
* **Secure data exchange** with authentication and encryption.

### **Key REST API Calls**

Azure Storage SFTP doesn't expose REST APIs directly but works through:

1. **List Files in SFTP-Enabled ADLS Container**

http

CopyEdit

GET https://<storageaccount>.dfs.core.windows.net/<container>?resource=filesystem

1. **Upload a File via SFTP**

bash

CopyEdit

sftp user@<storageaccount>.blob.core.windows.net  
put localfile.txt /remotepath/

1. **Delete a File via REST**

http

CopyEdit

DELETE https://<storageaccount>.blob.core.windows.net/<container>/<filename>

### **Performance**

* **Lower performance than AzCopy or WanDisco** for large datasets.
* Best suited for **small to medium file transfers**.
* Uses **SSH encryption**, which may add **CPU overhead**.

## **4. DistCp (Distributed Copy)**

### **Overview**

DistCp is a Hadoop-based tool designed for **parallel data movement** across HDFS clusters and cloud storage.

### **Use Cases**

* Moving **large-scale HDFS data** from on-prem to **Azure Data Lake Storage Gen2**.
* **Migrating petabyte-scale workloads** in a distributed fashion.

### **Key REST API Calls**

DistCp interacts with HDFS and Azure via WebHDFS REST APIs:

1. **Initiate Copy Job**

bash

CopyEdit

hadoop distcp hdfs://source-cluster/ hdfs://destination-cluster/

1. **Copy from HDFS to ADLS Gen2**

bash

CopyEdit

hadoop distcp -Dfs.azure.account.key.<storageaccount>.dfs.core.windows.net=<access\_key> hdfs://onprem/data [abfs://container@storageaccount.dfs.core.windows.net/data](mailto:abfs://container@storageaccount.dfs.core.windows.net/data)

1. **Check Job Status**

http

CopyEdit

GET http://<namenode>:50070/webhdfs/v1/path?op=GETFILESTATUS

### **Performance**

* **Highly scalable**—ideal for **terabyte-to-petabyte-scale migrations**.
* **MapReduce-based parallelization** for **efficient large data transfers**.
* **Slower than WanDisco for real-time sync** but great for **batch transfers**.

## **Performance & Feature Comparison Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Feature** | **AzCopy** | **WanDisco** | **SFTP** | **DistCp** |
| **Best for** | Large file & bulk transfers | Real-time sync & replication | Secure file transfers | Large-scale HDFS migration |
| **Transfer Speed** | **High** (multi-threaded) | **Very High** (real-time) | **Moderate** (SSH overhead) | **High** (distributed) |
| **Incremental Transfer** | ✅ Yes | ✅ Yes (real-time) | ❌ No | ✅ Yes |
| **Parallelism** | ✅ Yes | ✅ Yes | ❌ No | ✅ Yes |
| **Resume Support** | ✅ Yes | ✅ Yes | ✅ Yes | ✅ Yes |
| **Security** | ✅ TLS | ✅ Enterprise-grade | ✅ SSH | ✅ Kerberos & IAM |
| **Ease of Use** | ✅ Easy CLI | ❌ Complex | ✅ Simple | ❌ Requires Hadoop |

## **Conclusion**

* **For bulk file movement:** Use **AzCopy** for **high-speed**, large-scale transfers.
* **For real-time, continuous migration:** Use **WanDisco LiveData Migrator**.
* **For secure legacy file transfers:** Use **SFTP** on ADLS Gen2.
* **For large HDFS migrations:** Use **DistCp** for distributed, high-speed transfers.

Here are detailed scripts for moving data from on-premises to Azure using various CLI tools:

## **1. Using Azure CLI**

Azure CLI provides commands to upload files to Azure Blob Storage or ADLS Gen2.

### **Upload a File to Azure Blob Storage**

sh

CopyEdit

# Login to Azure  
az login  
  
# Set the storage account details  
STORAGE\_ACCOUNT\_NAME="mystorageaccount"  
CONTAINER\_NAME="mycontainer"  
LOCAL\_FILE\_PATH="/path/to/local/file.txt"  
REMOTE\_BLOB\_NAME="file.txt"  
  
# Upload the file  
az storage blob upload --account-name $STORAGE\_ACCOUNT\_NAME \  
 --container-name $CONTAINER\_NAME \  
 --name $REMOTE\_BLOB\_NAME \  
 --file $LOCAL\_FILE\_PATH \  
 --auth-mode login

### **Upload a Directory to Azure Data Lake Storage Gen2**

sh

CopyEdit

az storage fs directory create --account-name $STORAGE\_ACCOUNT\_NAME \  
 --file-system $CONTAINER\_NAME \  
 --path "mydirectory"  
  
az storage blob upload-batch --account-name $STORAGE\_ACCOUNT\_NAME \  
 --destination $CONTAINER\_NAME \  
 --source "/path/to/local/directory"

## **2. Using Databricks CLI**

Databricks CLI allows you to copy files from on-prem to Databricks FileStore.

### **Upload a File to Databricks FileStore**

sh

CopyEdit

# Configure Databricks CLI authentication  
databricks configure --host https://<databricks-instance> --token <your-access-token>  
  
# Upload a local file to Databricks FileStore  
databricks fs cp /path/to/local/file.txt dbfs:/mnt/mycontainer/file.txt

### **Upload a Directory to Databricks DBFS**

sh

CopyEdit

databricks fs cp --recursive /path/to/local/directory dbfs:/mnt/mycontainer/

### **Mount Azure Blob Storage or ADLS Gen2 in Databricks**

python

CopyEdit

dbutils.fs.mount(  
 source="wasbs://mycontainer@mystorageaccount.blob.core.windows.net",  
 mount\_point="/mnt/mycontainer",  
 extra\_configs={"fs.azure.account.key.mystorageaccount.blob.core.windows.net": "your-access-key"}  
)

## **3. Using Terraform**

Terraform automates infrastructure and data deployment to Azure.

### **Terraform Script to Upload Data to Azure Blob Storage**

hcl

CopyEdit

provider "azurerm" {  
 features {}  
}  
  
resource "azurerm\_resource\_group" "rg" {  
 name = "myResourceGroup"  
 location = "East US"  
}  
  
resource "azurerm\_storage\_account" "storage" {  
 name = "mystorageaccount"  
 resource\_group\_name = azurerm\_resource\_group.rg.name  
 location = azurerm\_resource\_group.rg.location  
 account\_tier = "Standard"  
 account\_replication\_type = "LRS"  
}  
  
resource "azurerm\_storage\_container" "container" {  
 name = "mycontainer"  
 storage\_account\_name = azurerm\_storage\_account.storage.name  
}  
  
resource "null\_resource" "upload\_file" {  
 provisioner "local-exec" {  
 command = "az storage blob upload --account-name ${azurerm\_storage\_account.storage.name} --container-name ${azurerm\_storage\_container.container.name} --file /path/to/local/file.txt --name file.txt --auth-mode login"  
 }  
}

### **Apply the Terraform Script**

sh

CopyEdit

terraform init  
terraform apply -auto-approve

## **4. Using AzCopy**

AzCopy is a high-performance CLI tool for moving data to Azure.

### **Upload a File**

sh

CopyEdit

azcopy copy "/path/to/local/file.txt" "<https://mystorageaccount.blob.core.windows.net/mycontainer/file.txt?your-SAS-token>"

### **Upload a Directory**

sh

CopyEdit

azcopy copy "/path/to/local/directory" "<https://mystorageaccount.blob.core.windows.net/mycontainer>" --recursive

### **Sync On-Prem Directory with Azure**

sh

CopyEdit

azcopy sync "/path/to/local/directory" "<https://mystorageaccount.blob.core.windows.net/mycontainer>" --recursive

## **5. Using Rsync (For Azure VM)**

If you're transferring data from an on-prem server to an Azure VM, rsync can be used.

### **Sync a Local Directory to an Azure VM**

sh

CopyEdit

rsync -avz -e "ssh -i /path/to/private-key.pem" /path/to/local/directory/ [azureuser@myvm.eastus.cloudapp.azure.com:/home/azureuser/data/](mailto:azureuser@myvm.eastus.cloudapp.azure.com:/home/azureuser/data/)

## **6. Using SFTP (For ADLS Gen2)**

Azure now supports SFTP on ADLS Gen2.

### **Upload a File to ADLS Gen2 Using SFTP**

sh

CopyEdit

sftp [myuser@my-storage-account.blob.core.windows.net](mailto:myuser@my-storage-account.blob.core.windows.net)put /path/to/local/file.txt /mycontainer/file.txt

## **Comparison of CLI Tools for Data Movement**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Tool** | **Best For** | **Speed** | **Ease of Use** | **Parallelism** |
| **Azure CLI** | Small to medium file transfers | Fast | Easy | No |
| **Databricks CLI** | Moving files to Databricks | Fast | Easy | No |
| **Terraform** | Automating data deployment | Moderate | Moderate | No |
| **AzCopy** | Large-scale bulk transfers | Very Fast | Moderate | Yes |
| **Rsync** | Syncing to Azure VMs | Fast | Moderate | Yes |
| **SFTP** | Secure transfers to ADLS | Slow | Easy | No |

Yes! Below is a **Databricks CLI script** that transfers data from on-premises to **Databricks FileStore (DBFS)** without mounting a storage account.

### **1. Prerequisites**

* Install **Databricks CLI** if you haven't already:

sh

CopyEdit

pip install databricks-cli

* Configure Databricks CLI authentication:

sh

CopyEdit

databricks configure --host https://<databricks-instance> --token <your-access-token>

### **2. Upload a File to DBFS**

sh

CopyEdit

databricks fs cp /path/to/local/file.txt dbfs:/FileStore/mydata/file.txt

### **3. Upload a Directory to DBFS**

sh

CopyEdit

databricks fs cp --recursive /path/to/local/directory dbfs:/FileStore/mydata/

### **4. List Uploaded Files in DBFS**

sh

CopyEdit

databricks fs ls dbfs:/FileStore/mydata/

### **5. Delete a File from DBFS**

sh

CopyEdit

databricks fs rm dbfs:/FileStore/mydata/file.txt

### **6. Download a File from DBFS to Local Machine**

sh

CopyEdit

databricks fs cp dbfs:/FileStore/mydata/file.txt /path/to/local/destination/

## **How to Access Uploaded Files in Databricks**

* Inside a **Databricks notebook**, you can read the uploaded file using:

python

CopyEdit

df = spark.read.text("dbfs:/FileStore/mydata/file.txt")  
display(df)

* Or download it via a web browser using:

perl

CopyEdit

https://<databricks-instance>/files/mydata/file.txt

You’re giving feedback on a new version of ChatGPT.

Which response do you prefer? Responses may take a moment to load.

ChatGPT

Response 1

Memory updated

Since **DBFS and storage account mounting are not allowed**, the best way to move data from on-prem to Databricks is by using **Databricks REST API** with Databricks CLI to directly upload files into a **cluster-local storage** (e.g., /tmp or /mnt) or using external storage like ADLS via service credentials.

## **Option 1: Upload Data to Cluster Local Storage (/tmp) Using REST API**

Since Databricks CLI doesn't allow direct uploads outside DBFS, use **Databricks REST API**.

### **1. Upload a File to a Databricks Cluster**

Use curl or Invoke-RestMethod in PowerShell to upload a file to **the /tmp directory** on a Databricks cluster.

#### **Bash (Linux/Mac)**

sh

CopyEdit

# Set variables  
DATABRICKS\_HOST="https://<your-databricks-instance>"  
DATABRICKS\_TOKEN="<your-personal-access-token>"  
CLUSTER\_ID="<your-cluster-id>"  
LOCAL\_FILE\_PATH="/path/to/local/file.csv"  
  
# Upload the file  
curl -n -X POST -H "Authorization: Bearer $DATABRICKS\_TOKEN" \  
 -F "file=@$LOCAL\_FILE\_PATH" \  
 "$DATABRICKS\_HOST/api/2.0/dbfs/upload?path=/tmp/file.csv&overwrite=true"

#### **PowerShell (Windows)**

powershell

CopyEdit

$DATABRICKS\_HOST = "https://<your-databricks-instance>"  
$DATABRICKS\_TOKEN = "<your-personal-access-token>"  
$CLUSTER\_ID = "<your-cluster-id>"  
$LOCAL\_FILE\_PATH = "C:\path\to\local\file.csv"  
  
Invoke-RestMethod -Uri "$DATABRICKS\_HOST/api/2.0/dbfs/upload?path=/tmp/file.csv&overwrite=true" `  
 -Headers @{Authorization=("Bearer " + $DATABRICKS\_TOKEN)} `  
 -Method Post `  
 -InFile $LOCAL\_FILE\_PATH `  
 -ContentType "multipart/form-data"

### **2. Access the File in Databricks Notebook**

python

CopyEdit

df = spark.read.csv("/tmp/file.csv", header=True)  
display(df)

## **Option 2: Use Azure Blob Storage (Without Mounting)**

Instead of **mounting**, use **temporary credentials (SAS token or Service Principal)** to access Azure Blob Storage.

### **1. Upload Data from On-Prem to Azure Blob Storage Using Azure CLI**

sh

CopyEdit

az storage blob upload --account-name mystorageaccount \  
 --container-name mycontainer \  
 --name file.csv \  
 --file /path/to/local/file.csv \  
 --auth-mode login

### **2. Read File in Databricks Without Mounting**

python

CopyEdit

spark.conf.set("fs.azure.account.auth.type.mystorageaccount.dfs.core.windows.net", "SAS")  
spark.conf.set("fs.azure.sas.token.provider.mystorageaccount.dfs.core.windows.net", "<your-sas-token>")  
  
df = spark.read.csv("abfss://mycontainer@mystorageaccount.dfs.core.windows.net/file.csv", header=True)  
display(df)

## **Best Approach Based on Your Constraints**

|  |  |  |
| --- | --- | --- |
| **Approach** | **Pros** | **Cons** |
| **Upload to Cluster /tmp using REST API** | No DBFS or Mounting required | Limited storage, resets when cluster restarts |
| **Upload to Azure Blob Storage & Read in Databricks** | Scalable & persistent storage | Requires Azure credentials (SAS token or Service Principal) |

Since **DBFS and mounting are not allowed**, you can use **Databricks REST API** to upload files directly into a Databricks workspace. Below is the approach using the **Databricks CLI and REST API**.

## **1. Using Databricks CLI to Upload Files to the Workspace**

Since DBFS is not an option, you can upload files to the Databricks **workspace (Workspace Files API)**.

### **Prerequisites**

* Ensure the **Databricks CLI** is installed:

sh

CopyEdit

pip install databricks-cli

* Authenticate with your **Databricks instance**:

sh

CopyEdit

databricks configure --host https://<databricks-instance> --token <your-access-token>

### **Upload a File to the Workspace**

sh

CopyEdit

databricks workspace import /path/to/local/file.txt /Workspace/MyFolder/file.txt --format AUTO

### **Upload a Directory to the Workspace**

sh

CopyEdit

databricks workspace import\_dir /path/to/local/directory /Workspace/MyFolder/

### **List Files in the Workspace**

sh

CopyEdit

databricks workspace ls /Workspace/MyFolder/

### **Delete a File**

sh

CopyEdit

databricks workspace rm /Workspace/MyFolder/file.txt

## **2. Using Databricks REST API to Upload Files**

If Databricks CLI is not an option, you can use the REST API.

### **Upload a File Using REST API**

sh

CopyEdit

curl -n -X POST https://<databricks-instance>/api/2.0/workspace/import \  
-H "Authorization: Bearer <your-access-token>" \  
-H "Content-Type: multipart/form-data" \  
-F path="/Workspace/MyFolder/file.txt" \  
-F format="AUTO" \  
-F file=@/path/to/local/file.txt

### **List Files in the Workspace**

sh

CopyEdit

curl -n -X GET https://<databricks-instance>/api/2.0/workspace/list \  
-H "Authorization: Bearer <your-access-token>" \  
-d '{  
 "path": "/Workspace/MyFolder/"  
}'

## **3. Accessing the Files in a Databricks Notebook**

Once uploaded, you can access the files in a Databricks notebook:

python

CopyEdit

df = spark.read.text("file:/Workspace/MyFolder/file.txt")  
display(df)

### **Alternative Approach: Direct Upload to Azure Storage from On-Prem**

Since Databricks is running on **Azure**, it's often best to upload data to **Azure Blob Storage** or **ADLS Gen2** first. You can then read it in Databricks **without mounting**, using a SAS token or service principal authentication.

Example:

python

CopyEdit

df = spark.read.csv("abfs://container@mystorageaccount.dfs.core.windows.net/mydata/file.csv", header=True)  
display(df)