Lab 11: Accessing ADLS Using a SAS Token (Beginner-Friendly, Step-by-Step)

Author: Dr. Sandeep Kumar Sharma

Learning Objective

This lab teaches beginners how to access files in an Azure Data Lake Storage (ADLS) / Azure Blob Storage container using a SAS (Shared Access Signature) token from within Azure Databricks. You will learn how to generate a SAS token, use it directly to read files, configure Spark to use the SAS token, and optionally mount the storage to DBFS.

Learning Outcome

By the end of this lab you will be able to: - Understand what a SAS token is and when to use it - Generate a SAS token with appropriate permissions from the Azure Portal - Read data from ADLS/Blob using a SAS token (direct URL method) - Configure Spark to use the SAS token and read using `wasbs://` path - Mount and unmount the storage container using the SAS token (POC only) - Understand the security implications and best practices

> ⚠️**Security reminder:** SAS tokens grant direct access to storage. They can expire and should be treated like secrets. **Do not** hard-code SAS tokens in production notebooks. Use Azure Key Vault, Service Principal, or Managed Identity for production workflows.

Prerequisites

1. You must have an Azure subscription and access to the target storage account.
2. You must have permission to generate SAS tokens for the storage container (Storage Blob Data Contributor or owner role usually required in Portal).
3. A CSV file named `employee.csv` exists inside container `raw` in the storage account `yourstorageaccount`.

Example file URL (no SAS yet):

```
https://yourstorageaccount.blob.core.windows.net/raw/employee.csv
```

Part A — Concept: What is a SAS Token? (Short & Simple)

A **SAS (Shared Access Signature)** token is a URI component that grants limited access to storage resources without exposing the account key. With SAS you can restrict: - Permissions (Read, Write, List, Delete) - Time window (start and expiry time) - IP range and protocol (HTTPS)

Think of it as a temporary key that lets someone download or upload files for a limited time.

Part B — Generate a SAS Token (Azure Portal)

Follow these GUI steps (show them to beginners slowly): 1. Open the **Azure Portal** and go to your **Storage Account** (`yourstorageaccount`). 2. In the left menu choose **Containers** and open the container `raw` . 3. Click on the three dots `...` next to the container and choose **Generate SAS** (or go to "Shared access signature" under "Security + networking"). 4. Set the permissions you need — for this lab pick **Read** and **List**. 5. Choose an expiry time (for lab use something like 1 day from now). 6. Optionally restrict allowed IPs or protocols (HTTPS recommended). 7. Click **Generate SAS token and URL**. 8. Copy the **SAS token** (it usually begins with `?sv=...`) or copy the full **URL with SAS**.

Now you have a SAS token like:

```
?
sv=2024-...&ss=b&srt=sco&sp=rl&se=2025-11-25T15:00:00Z&st=2025-11-24T08:00:00Z&spr=https&sig=abc1
```

Part C — Method 1: Direct URL Read (Quick & Easy)

This method appends the SAS token to the file HTTPS URL and uses Spark to read it directly. It is the simplest for beginners.

Step C.1 — Prepare Variables (Notebook cell)

```
# Replace these placeholders with your actual values from Azure
storage_account = "yourstorageaccount"
container = "raw"
file_name = "employee.csv"
# Paste the SAS token you copied from Azure Portal (including leading ?)
sas_token = "?sv=...YOUR_SAS_TOKEN..."

# Construct the full URL
```

```
file_url_with_sas = f"https://{storage_account}.blob.core.windows.net/
{container}/{file_name}{sas_token}"
print(file_url_with_sas)  # for debugging only
```

Step C.2 — Read CSV directly from HTTPS URL

```
# Read CSV directly from URL with SAS
df = spark.read.csv(file_url_with_sas, header=True, inferSchema=True)
display(df)
df.printSchema()
```

Notes: - This works well for reading single files or small datasets. - If you have many files or large data, prefer configuring Spark with the SAS token or mounting the container.

Part D — Method 2: Configure Spark with SAS Token (Preferred for Multiple Files)

Instead of embedding SAS in every URL, configure Spark so you can use `wasbs://` (Blob) paths. This is cleaner when working with many files in the container.

Step D.1 — Prepare the SAS token string

Spark expects the SAS to be stored without the leading `?` in config. We'll remove it just in case.

```
# ensure SAS token does not start with '?'
clean_sas = sas_token[1:] if sas_token.startswith('?') else sas_token
```

Step D.2 — Set Spark config

```
spark.conf.set(
    f"fs.azure.sas.{container}.{storage_account}.blob.core.windows.net",
    clean_sas
)
```

This tells Spark: "If any `wasbs://raw@yourstorageaccount.blob.core.windows.net/...` path is used, attach this SAS token."

Step D.3 — Read using wasbs path

```
wasbs_path = f"wasbs://{container}@{storage_account}.blob.core.windows.net/
{file_name}"

df2 = spark.read.csv(wasbs_path, header=True, inferSchema=True)
display(df2)
df2.printSchema()
```

Notes: - `wasbs://` is the Hadoop-compatible URI for Azure Blob Storage. Spark uses the configuration key we set to sign requests. - For ADLS Gen2 (Hierarchical Namespace), the URI scheme `abfss://` is used; configuration keys differ slightly (covered in the Optional Tips section).

Part E — Method 3: Mount the Container in DBFS Using SAS (POC Only)

Mounting makes the container look like `/mnt/raw` and is handy in notebooks. **Only do this in POC/ learning scenarios.**

Step E.1 — Define mount point and configs

```
mount_point = "/mnt/raw_sas"
source = f"wasbs://{container}@{storage_account}.blob.core.windows.net"
extra_configs = {f"fs.azure.sas.{container}.
{storage_account}.blob.core.windows.net": clean_sas}
```

Step E.2 — Mount (run once)

```
# Check if already mounted to avoid errors
mounts = [m.mountPoint for m in dbutils.fs.mounts()]
if mount_point not in mounts:
    dbutils.fs.mount(
        source=source,
        mount_point=mount_point,
        extra_configs=extra_configs
    )
    print("Mounted successfully")
else:
    print("Mount point already exists")
```

Step E.3 — List files and read

```
display(dbutils.fs.ls(mount_point))

# Read from mount
mounted_df = spark.read.csv(f"{mount_point}/{file_name}", header=True,
inferSchema=True)
display(mounted_df)
```

Step E.4 — Unmount when done (optional cleanup)

```
# Only unmount if you want to remove the mount
if mount_point in mounts:
    dbutils.fs.unmount(mount_point)
    print("Unmounted")
```

Notes: - Mounted SAS tokens are persisted in the cluster until unmounted or the token expires. If the token expires the mount will fail to read. - Avoid mounting in production; prefer managed identities or service principals.

Part F — ADLS Gen2 (abfss) tip — short note

If your account uses ADLS Gen2 with hierarchical namespace, you would typically use `abfss://<container>@<account>.dfs.core.windows.net/<path>` as the path. Using SAS with `abfss` may require slightly different configuration keys like `fs.azure.sas.<container>.<account>.dfs.core.windows.net`. If you need this, tell me and I will add an explicit abfss example.

Part G — Security Best Practices (for Beginners to remember)

1. **Don't store SAS in cleartext** inside notebooks pushed to repos. Use secrets or Key Vault.
2. **Set a short expiry** for test SAS tokens.
3. **Limit permissions** (give only read if you only need read access).
4. **Restrict IP** addresses if possible during generation.
5. **Rotate** tokens periodically.

Practice Tasks (Small Exercises)

1. Generate a SAS token with only **Read** and **List** permissions and read the file using **Method 1**.
2. Configure Spark with the SAS (Method 2) and read multiple files from the container using a wildcard like `wasbs://raw@yourstorageaccount.blob.core.windows.net/*.csv`.
3. Mount the container (Method 3) and practice listing, reading, and unmounting.

Troubleshooting Tips

- If you see `403 Forbidden` or authentication errors, verify the SAS token has not expired and has correct permissions.
- If mount fails, check for existing mount points and token expiry.
- For `abfss` paths, confirm whether your storage account has HNS enabled.

End of Lab 11

You have learned how to use SAS tokens to access ADLS/Blob storage from Databricks with three practical methods, along with security best practices and troubleshooting. If you want, I will now create **Lab 12 – Accessing ADLS Using Service Principal (Production)** in the canvas.