AZURE DATBRICKS AND AZURE SYNAPSE ANALYTICS

Load data from Azure Container to Azure Synapse Table

ABSTRACT

This document describes how to load data present in Azure Blob Storage to Azure Synapse tables using Azure Python SDK.

Ashish Sinha

Senior Data Engineer

Table of Contents

1. Introduction:

The requirement is to push data from all the csv files present inside a container folder to Azure Synapse Table. The container heirachy is landing-area/ folder-name/ inbound. The file are present in Inbound folder and after being processed needs to be archived in the landing-area/ folder-name/ archive.

2. Prerequisites:

- Azure Subscription
- Azure Storage Account
- Azure Key Vault
- Azure Databricks
- Azure Databricks Secret Scope
- Azure Synapse Analytics

1. To create your free Azure account please go to this link

2. To see how to create your Azure resource group, Azure Storage account, Azure Databricks workspace and Azure Key-vault please follow the steps here.

   Note: *In the link above, we are taught how to mount your container in dbfs which is not what we are going to do here. We will be using Python SDK to connect to Azure Blob Storage and load data into Azure Synapse.*

3. To read the official documentation on how to connect to Azure Synapse Analytics using Databricks using Scala or SQL you can follow this link.

   Note: *Don't forget to Set **Allow access to Azure services** to **ON,** on the firewall pane of the Azure Synapse server through Azure portal.*

3. Azure Blob Storage

Connect to Azure Blob Storage

Import Libraries:

First of all, we will import all the required libraries to connect to azure storage blob.

```
# Import Required Libraries

from azure.storage.blob import BlobServiceClient
from pyspark.sql.functions import *
from azure.storage.blob import ContainerClient
```

Details to Access Blob Storage:

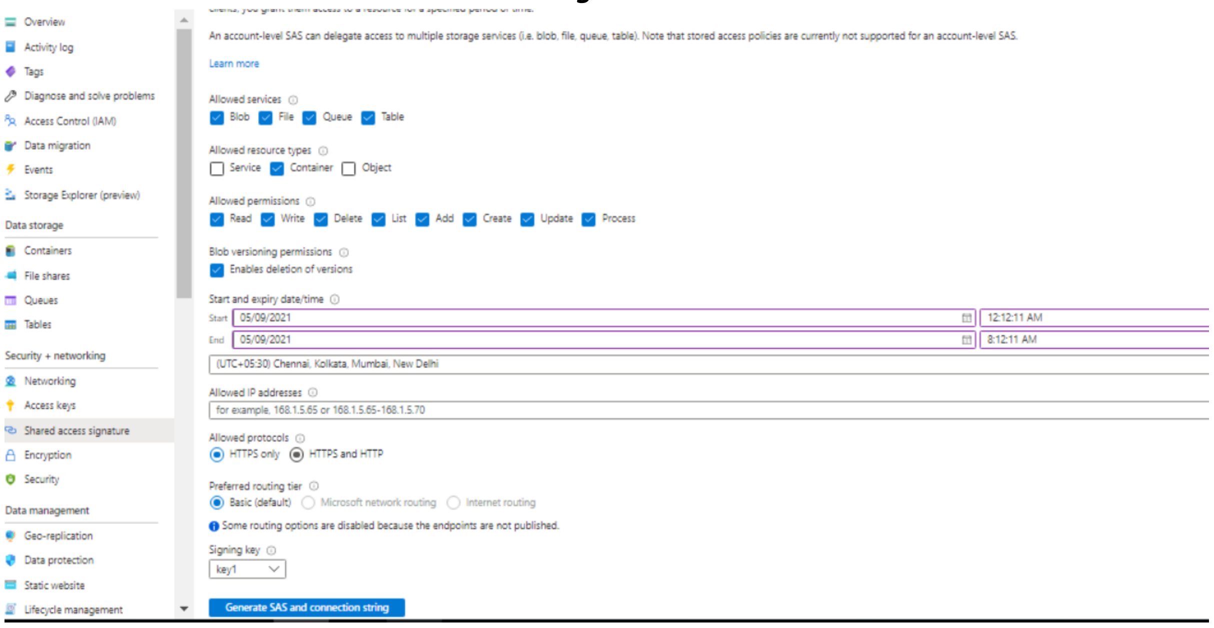
We will then create some variables with the required information to connect to Azure Blob Storage. This includes storage account name, container name, SAS token, wasbs path, access keys.

```
blob_account_name = "<your-storage-account-name>"
blob_container_name = "<your-container-name>"
blob_folder_name = "<your-folder-in-container>"
blob_relative_path = blob_folder_name + "/Inbound"
blob_sas_token = r”<your-storage-account-sas-token>”
blobwasbspath = "<your wasbs path for blob>"

# For e.g., wasbs://<blob_container_name>
@blob_account_name.blob.core.windows.net/

blob_account_key = "<your_blob_account_key>"

# wasbs path for a blob and spark conf set
wasbs_path = 'wasbs://%s@%s.blob.core.windows.net/%s' % (blob_container_name,
blob_account_name, blob_relative_path)

spark.conf.set('fs.azure.sas.%s.%s.blob.core.windows.net' % (blob_container_name,
blob_account_name), blob_sas_token)
print('Remote blob path: ' + wasbs_path)

#connection string for blob
connection_string =
"DefaultEndpointsProtocol=https;AccountName=%s;AccountKey=%s;EndpointSuffix=core.wi
ndows.net" % (blob_account_name, blob_account_key)
```

```
# Set up the Blob storage account access key in the notebook session conf.
spark.conf.set(
  "fs.azure.account.key. <storage-account-name>. blob.core.windows.net",
  "<blob_account_key>")
```

Steps to create Shared access signature: -

1. Go to your storage account
2. Go to Shared Access Signature in the Left Pane.
3. Click on "***Container***" in ***Allowed Resources Types***
4. Select the Required Permissions
5. Select the start and expiry of token
6. Click on ***Generate SAS and Connection String.***



An account-level SAS can delegate access to multiple storage services (i.e. blob, file, queue, table). Note that stored access policies are currently not supported for an account-level SAS.

Allowed services: ✓ Blob ✓ File ✓ Queue ✓ Table

Allowed resource types: Service [✓ Container] Object

Allowed permissions: ✓ Read ✓ Write ✓ Delete ✓ List ✓ Add ✓ Create ✓ Update ✓ Process

Blob versioning permissions: ✓ Enables deletion of versions

Start and expiry date/time
Start 05/09/2021 12:12:11 AM
End 05/09/2021 8:12:11 AM
(UTC+05:30) Chennai, Kolkata, Mumbai, New Delhi

Allowed IP addresses: for example, 168.1.5.65 or 168.1.5.65-168.1.5.70

Allowed protocols: [✓ HTTPS only] HTTPS and HTTP

Preferred routing tier: [✓ Basic (default)] Microsoft network routing Internet routing

Some routing options are disabled because the endpoints are not published.

Signing key: key1

7. After that it will generate
   a. Connection String
   b. SAS Token: - The query string that includes all of the information required to authenticate the SAS, as well as to specify the service, resource, and permissions available for access, and the time interval over which the signature is valid. To construct an account SAS URL, append the SAS token to the URL for a storage service, or use one of the URLs below.
   c. Blob service SAS URL
   d. File service SAS URL
   e. Queue service SAS URL
   f. Table service SAS URL

Steps to get Access Keys: -

1. In the same Left Pane, click on Access Keys
2. By default, there are 2 keys present. You can use either one

4. Azure Synapse Details

Connect to Azure Synapse Workspace

Now we will connect to Azure Synapse workspace. For this we would require account name, the default port and database name.

```
#connection details for synapse analytics
dwServer = "<your-synapse-workspace>. sql.azuresynapse.net"
dwPort = "1433"
dwDBName = "<db name>"

#Retrieve sql username and password from key vault scope
dwUserName = dbutils.secrets.get(scope = "key-vault-secrets", key = "sqluser")
dwPassword = dbutils.secrets.get(scope = "key-vault-secrets", key = "sqlpassword")

#Create link to connect with the Synapse via JDBC url
tempDir = "wasbs://" + blob_container_name + "@" + blob_account_name +
".blob.core.windows.net/" + blob_folder_name
dwUrl =
"jdbc:sqlserver://"+dwServer+":"+dwPort+";database="+dwDBName+";user="+dwUserNam
e+";password="+dwPassword+";encrypt=true;trustServerCertificate=true;hostNameInCertifi
cate=*.sql.azuresynapse.net;loginTimeout=30;"
```

5. List and Read Files in container

Now we will list the files present in storage account and read it into PySpark data frame. This data frame would be used to load data into synapse table

```
#Container connection to access blobs in a container
container = ContainerClient.from_connection_string(conn_str=connection_string,
container_name=blob_container_name)

#list of all files in a container
files = []
count = 0
blob_list = container.list_blobs()
```

```
for blob in blob_list:
   files.append(blob.name)
   count=count+1
print("Files to be processed are: -")
print(files)
```

6. Load Data to Azure Synapse

Here we will load the data frame into table. We are using overwrite to recreate and load table if it exists. We can also use append to load data into existing table and it will append with existing records. But for this we need to make sure that the data frame structure and target table structure is same otherwise the code will fail

.

```
#read the csv files and populate the corresponding tables
table_names = ["I","PR","RS","TS","ASGN_BLS","PG_BS","TS_BS","RESOURCES","TmSet"]

#loop to table names and process the csv files to Synapse Database
for name in table_names:
  for file in files:
    if name in file:
      print("File name:- {}".format(file))
      df = spark.read.csv(wasbs_path + "/"+file, header = 'true')

      # Add Load Date column in the data frame as current timestamp
      df = df.withColumn("LOAD_DATE",current_timestamp())

      #Convert date columns in file to datatype datetime
      datecols = [x for x in df.columns if x.endswith('Date')]
      for col in datecols:
        df = df.withColumn(col,to_timestamp(df[col], 'yyyy-MM-dd HH:mm:ss'))

      #write the data from the file to table
      df.write \
        .format("com.databricks.spark.sqldw") \
        .option("url", dwUrl) \
        .option("forwardSparkAzureStorageCredentials", "true") \
        .option("dbTable", "dbo."+name+"_RAW ") \
        .option("tempDir", tempDir) \
        .option("truncate","true") \
        .option("maxStrLength", "4000" ) \
        .mode("overwrite") \

      print("File {} is loaded to synpase".format(file))
```

7. Functions to Copy and Delete Blob

Here we have function to copy blob from one location to another and then delete it. This basically works as Move Blob from one location to another.

```
# Function to Copy blob from one folder to another
def copy_blob(account_name,container,folder,file,con_str):
    status = None
    blob_service_client = BlobServiceClient.from_connection_string(con_str)
    source_blob = "https://%s.blob.core.windows.net/%s/%s/Inbound/%s" %
(account_name,container,folder,file)
    copied_blob = blob_service_client.get_blob_client(container+"/"+folder+"/Archive",file)

    copied_blob.start_copy_from_url(source_blob)
    for i in range(10):
      props = copied_blob.get_blob_properties()
      status = props.copy.status
      print("Copy Status: "+status)
      if status == "success":
        break
      time.sleep(3)

    if status != "success":
      # if not finished after 30s, cancel the operation
      props = copied_blob.get_blob_properties()
      print(props.copy.status)
      copy_id = props.copy.id
      copied_blob.abort_copy(copy_id)
      props = copied_blob.get_blob_properties()
      print(props.copy.status)

# Function to delete blob
def delete_blob(container_client,path,file):
  full_path = path+"/"+file
  container_client.delete_blob(full_path)
  print("File {} has been deleted".format(file))
```