



AZURE DATABRICKS - SMARTSHEET TO SYNAPSE

Load Smartsheet data to Synapse Table

ABSTRACT

This document would describe how to use Azure Databricks to load smartsheet data to Azure Synapse using Smartsheet API

Ashish Sinha

Senior Data Engineer

Contents

Introduction	2
Requirement and Issue	2
Process Description.....	2
Code Steps and Process	3
1. Create ADBS widgets.....	3
2. Import Libraries and Connect to Smartsheet	3
3. Get Smartsheet Data and load it to Pandas Dataframe.....	4
4. Function to clean data and create Spark Dataframe from Pandas Dataframe.....	5
5. Load to Synapse	5
6. Running from Azure Data Factory.....	7

Introduction

This document would go through the process and code to connect to your smartsheet and load the data in it to Azure Synapse table for data analysis.

The scheduling and running the Databricks notebook would be done using parametrized Azure Data Factory pipeline.

Requirement and Issue

1. The requirement is pretty simple but the issue is sometimes the data in your smartsheet would have unwanted characters such as comma (,) that would break the rows into multiple rows if you convert it to csv and load it using Azure Data Factory.
2. I also tried using excel and load the data from it. The records in synapse were breaking into multiple rows and all the data was messed up.
3. PDF file would not have worked for this scenario.

Process Description

1. In Azure Databricks (ADBS), we would create a generic template that would be used to load any smartsheet data to any Azure Synapse table.
2. We would use widgets in notebook to pass parameters from ADF Pipeline.
3. Parameters would include Smartsheet Access Token, Sheet ID and Synapse Schema and Table name.
4. We would connect with Smartsheet API and Synapse using JDBC Connectors respectively.
5. We would dynamically extract columns and data from smartsheet and create a pandas data frame from it, which would be loaded to Synapse table.

The process to run ADBS notebook from ADF with parameter is described in the [link](#) here.

Code Steps and Process

1. Create ADBS widgets

Create required widgets so as to pass parameters from ADF pipeline.

The parameters would be Smartsheet Access Token, Smartsheet Sheet ID, Synapse Schema and Table Name.

```
#####  
## Get the Access token and Sheet Id from ADF Pipeline  
#####  
dbutils.widgets.text("access_token", "", "")  
access_token = dbutils.widgets.get("access_token")  
print ("Parameter Access token :")  
print (access_token)  
  
dbutils.widgets.text("sheetId", "", "")  
sheetId = dbutils.widgets.get("sheetId")  
print ("Parameter Sheet ID :")  
# Convert sheet ID to interger type  
sheetid = int(sheetId)  
print (sheetId)  
  
dbutils.widgets.text("tableSchema", "", "")  
tableSchema = dbutils.widgets.get("tableSchema")  
print ("Parameter Table Schema :")  
print (tableSchema)  
  
dbutils.widgets.text("tableName", "", "")  
tableName = dbutils.widgets.get("tableName")  
print ("Parameter Table Name :")  
print (tableName)  
  
table = tableSchema + "." + tableName  
print(table)
```

2. Import Libraries and Connect to Smartsheet

Import required libraries to be used and make a connection with smartsheet with the access token and sheet id passed from variable above.

```
#####  
## Import Required Libraries  
#####  
  
import smartsheet  
import pandas as pd  
# Initialize client  
smartsheet_client = smartsheet.Smartsheet(access_token =  
access_token)  
  
# Make sure we don't miss any errors  
smartsheet_client.errors_as_exceptions(True)  
  
#####  
## Get the sheet  
#####  
sheet = smartsheet_client.Sheets.get_sheet(sheetId)
```

3. Get Smartsheet Data and load it to Pandas Dataframe

```
#####
## Get Column Id and Name from sheet.columns
#####
column_id = [i.id for i in sheet.columns]
column_name = [i.title for i in sheet.columns]

#####
## Map column id and its corresponding name and present it in a
dataframe
#####
dict_columns = {"columnId":column_id, "Name":column_name}
df_columns = pd.DataFrame.from_dict(dict_columns)
print(df_columns.head())

#####
## created an empty list "rows" to store the cells data in sheet rows
after converting it to a dictionary. Append the dictionary data into
above list
#####
rows=[]
for i in sheet.rows:
    data = i.to_dict()
    rows.append(data["cells"])

#####
## create a dict with keys as the column id and value as an empty
list.
#####
data = {i:[] for i in column_id}
print(data)

#####
## Loop through data in rows. Each row in rows has column id and
value of the cell. If value is not present then it is to be
considered empty else fetch the value from row dictionary. Append
this value into the list of values for the key present in the data
dictionary.
#####
# rows is the list of dictionary.
for row in rows:
    for key in row:
        if 'value' not in key:
            value = None
        else:
            value = key['value']
            data[key['columnId']].append(value)
#print(sample_dict)

#####
## Load the data in dataframe and rename the columns
#####
df = pd.DataFrame(data)
df.columns = column_name
df.fillna("", inplace=True)
print(df.tail())
```

4. Function to clean data and create Spark Dataframe from Pandas Dataframe

```
#####  
## Function to find rows with data as None in all columns  
#####  
  
def cleaned_df(df):  
    for index, rows in df.iterrows():  
        row = []  
        for i in df.columns:  
            row.append(rows[i])  
            ##### create a set to get only unique values  
        row_set = set(row)  
        if len(row_set) == 1:  
            df.drop(index, inplace = True)  
    return df  
  
# call the function  
df = cleaned_df(df)  
print(df.tail())  
  
# replace unwanted characters from the columns and rename them  
cols =  
[i.replace(".", "").replace("(", "").replace(")", "").replace("/", "").re  
place("  
", "_").replace(",", "").replace(";", "").replace("{", "").replace("}", "  
").replace("\n", "").replace("\t", "").replace("=", "") for i in  
df.columns]  
df.columns = cols  
  
# convert column to string. This is so that it is easy to convert the  
df to sparkDF  
for i in df.columns:  
    if df[i].dtype == 'object':  
        df[i] = df[i].astype('str')  
print(df.tail())  
  
# convert pandas to sparkdf  
spark.conf.set("spark.sql.execution.arrow.enabled", "true")  
sparkDF=spark.createDataFrame(df)  
display(sparkDF)
```

5. Load to Synapse

Now finally load data to synapse

```
blob_account_name = ""  
blob_container_name = ""  
blob_folder_name = ""  
blob_account_key = ""
```

```

#connection details for synapse analytics
dwServer = ""
dwPort = "1433"
dwDBName = ""

#Retrieve sql username and password from key vault scope
dwUserName = dbutils.secrets.get(scope = "synapse-secret-scope", key =
"sqluser")
dwPassword = dbutils.secrets.get(scope = "synapse-secret-scope", 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=" + dw
wUserName + ";password=" + dwPassword + ";encrypt=true;trustServerCertificate=
true;hostNameInCertificate=*.sql.azure-synapse.net;loginTimeout=30;"

#connection string for blob
connection_string =
"DefaultEndpointsProtocol=https;AccountName=%s;AccountKey=%s;EndpointSuf
fix=core.windows.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.bgnesynapse.blob.core.windows.net",
    blob_account_key)

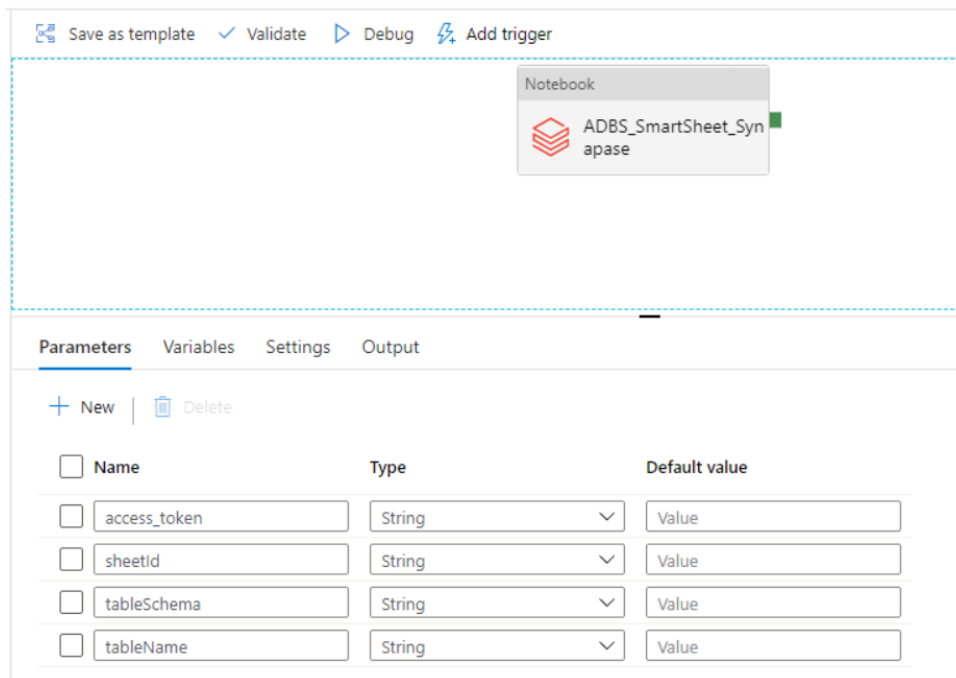
#####
## Write Dataframe to Synapse table
#####
sparkDF.write \
    .format("com.databricks.spark.sqldw") \
    .option("url", dwUrl) \
    .option("forwardSparkAzureStorageCredentials", "true") \
    .option("dbTable", table) \
    .option("tempDir", tempDir) \
    .option("truncate", "true") \
    .option("maxStrLength", "4000" ) \
    .mode("overwrite") \
    .save()

```

6. Running from Azure Data Factory

Here are the screenshots on how to run notebook from ADF pipeline.

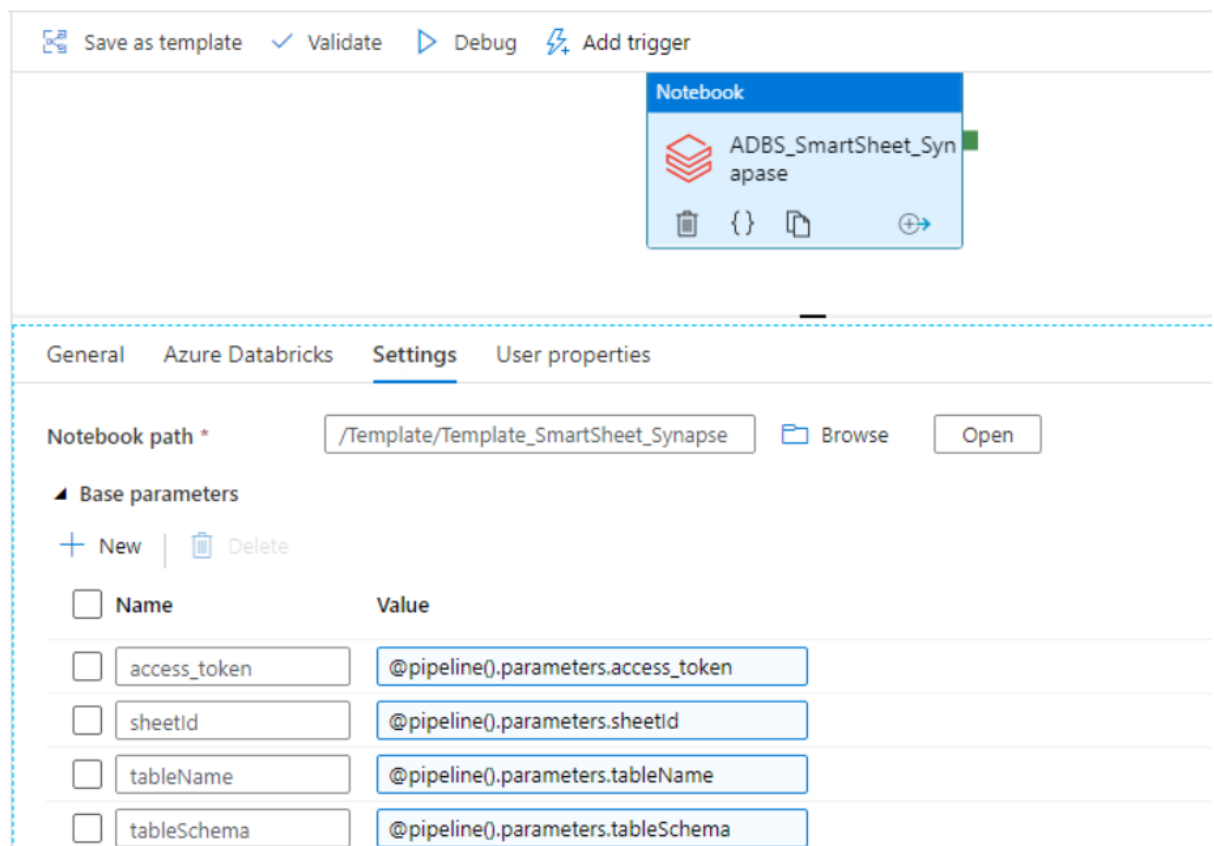
1. Parameters created in ADF Pipeline



The screenshot shows the 'Parameters' tab in the ADF Pipeline configuration. At the top, there are buttons for 'Save as template', 'Validate', 'Debug', and 'Add trigger'. Below these is a 'Notebook' card for 'ADBS_SmartSheet_Synapse'. The 'Parameters' tab is selected, showing a table with four parameters: 'access_token', 'sheetId', 'tableSchema', and 'tableName'. Each parameter is of type 'String' and has a default value of 'Value'.

Name	Type	Default value
access_token	String	Value
sheetId	String	Value
tableSchema	String	Value
tableName	String	Value

2. Passing there parameters in ADBS notebook from Base Parameters



The screenshot shows the 'Settings' tab in the ADBS Notebook configuration. At the top, there are buttons for 'Save as template', 'Validate', 'Debug', and 'Add trigger'. Below these is a 'Notebook' card for 'ADBS_SmartSheet_Synapse'. The 'Settings' tab is selected, showing the 'Notebook path' as '/Template/Template_SmartSheet_Synapse'. Under 'Base parameters', there is a table with four parameters: 'access_token', 'sheetId', 'tableName', and 'tableSchema'. Each parameter is mapped to a value starting with '@pipeline().parameters.'.

Name	Value
access_token	@pipeline().parameters.access_token
sheetId	@pipeline().parameters.sheetId
tableName	@pipeline().parameters.tableName
tableSchema	@pipeline().parameters.tableSchema