Korn Ferry Data Lake

Ver 1.0

June 2021

Contents

[1. Introduction 4](#_Toc75189275)

[1.1. Purpose 4](#_Toc75189276)

[1.2. Scope 4](#_Toc75189277)

[1.3. Out of Scope 4](#_Toc75189278)

[2. Target State Architecture 4](#_Toc75189279)

[2.1. Key Design Considerations 4](#_Toc75189280)

[2.2. Conceptual Architecture 4](#_Toc75189281)

[2.3. Data Flow Diagram 4](#_Toc75189282)

[3. KF Framework 4](#_Toc75189283)

[3.1. Data Ingestion Framework 4](#_Toc75189284)

[3.2. Key components 4](#_Toc75189285)

[3.3. ER Diagram 4](#_Toc75189286)

[3.4. Process Flow 4](#_Toc75189287)

[3.5. Data Validation 4](#_Toc75189289)

[3.6. Modularized Functions 4](#_Toc75189290)

[3.7. Restart ability 4](#_Toc75189291)

[3.8. Retry 4](#_Toc75189292)

[3.9. Logging Mechanism 4](#_Toc75189293)

[3.10. Alert and Notification 4](#_Toc75189294)

[3.11. Exception Handling 4](#_Toc75189295)

[4. Source System 4](#_Toc75189296)

[5. Data Loading Strategy 4](#_Toc75189297)

[5.1. Full Load 4](#_Toc75189298)

[5.1.1. ADF Job name: 4](#_Toc75189299)

[5.1.2. Command to run: 4](#_Toc75189300)

[5.1.3. Steps for Full Load: 4](#_Toc75189301)

[5.2. Delta Load (Incremental load) 4](#_Toc75189302)

[5.2.1. JOB NAME: 4](#_Toc75189303)

[5.2.2. Command to run: 4](#_Toc75189304)

[5.2.3. Steps For Delta Load: 4](#_Toc75189305)

[6. Naming & Coding Standards 5](#_Toc75189306)

[6.1. Coding Standards 5](#_Toc75189307)

[6.2. Table Names 5](#_Toc75189308)

[6.2.1. Data Ingestion Framework Tables 5](#_Toc75189309)

[6.2.2. Data Lake Layer Tables 5](#_Toc75189310)

[6.2.3. Delta Layer Tables 5](#_Toc75189311)

[6.3. Variable Names 5](#_Toc75189312)

[6.4. Function Names 5](#_Toc75189313)

[6.5. Script Names 5](#_Toc75189314)

[7. Retention Strategy 5](#_Toc75189315)

[8. Capacity Plan 5](#_Toc75189316)

[8.1. Hardware 5](#_Toc75189317)

[8.2. Software 5](#_Toc75189318)

[9. Git hub Process 5](#_Toc75189319)

[10. Job Orchestration (ADF) 5](#_Toc75189320)

[11. Spark Process 5](#_Toc75189321)

[11.1. Requirement 5](#_Toc75189322)

[11.2. Process Flow 5](#_Toc75189323)

[11.3. Data Used 5](#_Toc75189324)

[11.4. Step by step Solution 5](#_Toc75189325)

[12. Go Live Process (Dev-Ops) 5](#_Toc75189326)

[13. Run Book 5](#_Toc75189327)

[14. Maintenance 5](#_Toc75189328)

[15. Data Governance 5](#_Toc75189329)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Version No.*** | ***Date*** | ***Description*** | ***Author*** | ***Reviewed By*** |
| 1.0 | 21/06/2021 | Initial document | Prashant Rajguru |  |

# Introduction

## Purpose

The purpose of this project is to extract the data from Arya profiles source system and load the data into delta lake tables on top of Azure data lake storage gen 2 so that it can be used for further processing.

## Scope

Scope includes below points -

* Extraction of zip files from source system via ADF pipeline to azure data lake gen 2 storage
* Processing the extracted json files into tabular format and flattening the nested structure.
* Saving the processed data into delta lake tables to be stored on top of the azure data lake storage gen 2
* Monitoring of pipeline
* Logging

## Out of Scope

# Target State Architecture

## Key Design Considerations

* Full and Incremental zip data extraction – to allow flexibility to extract and copy only newly obtained zip files or to extract all zip files.
* Separate loading of each batch into separate delta table in the azure data lake storage.

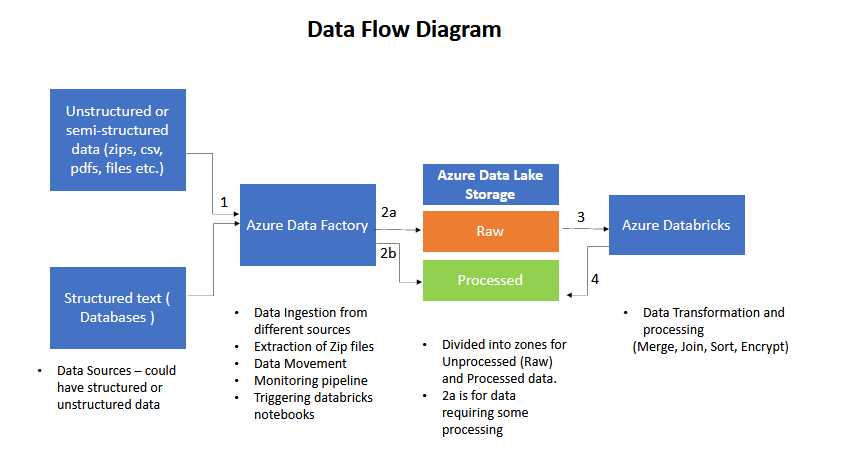
## Conceptual Architecture

Conceptual Architecture contains below steps

* + 1. Extraction of zip files present in the source location and copying of extracted data to ADLS gen 2 target location
    2. Moving the Source zip files so that they are not copied again in next batch
    3. Processing the extracted jsons batchwise, combining them together and flattening the nested structure and storing them in delta tables.

## Data Flow Diagram

## A high-level architecture with data flow diagram is provided below. In the case of Arya-profiles, the source data is present in zip files and it is fed into the Azure Data Factory after configuring linked services and data sources. It is loaded into ADLS gen2 via ADF into the extracted directory. Databricks notebook is later run on the extracted Jsons and it creates delta tables batchwise and saves them to ADLS Gen 2 Storage account.



# KF Framework

## Data Ingestion Framework

ADF pipeline is used for data ingestion and extraction of compressed data from azure blob storage to azure data lake storage gen 2. Databricks is later used for loading the data into delta tables.

## Key components

ADF Pipeline- zipextraction

Databricks Notebook-

## ER Diagram

## Process Flow



## Data Validation

## Modularized Functions

## Restart ability

## Retry

## Logging Mechanism

## Alert and Notification

## Exception Handling

# Source System

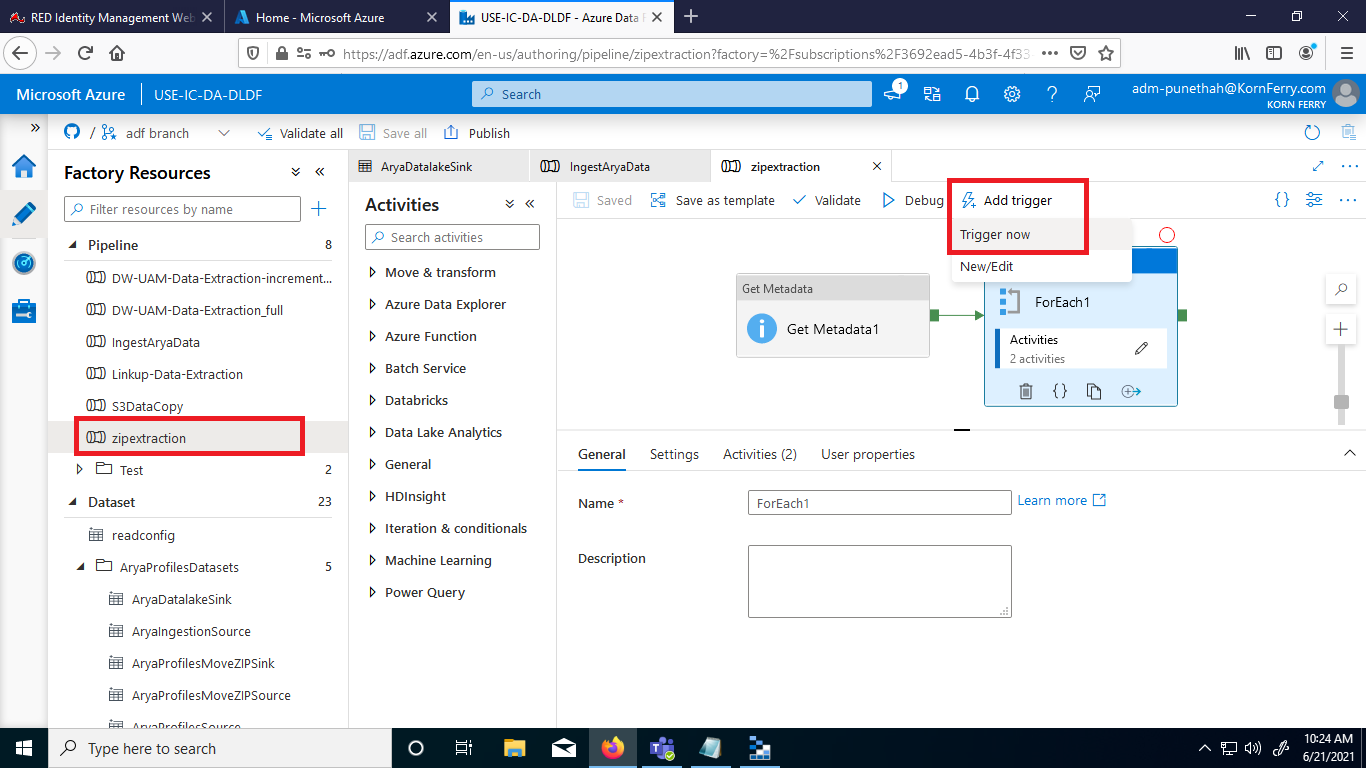
# Data Loading Strategy

## Full Load

## ADF Job name: zipextraction

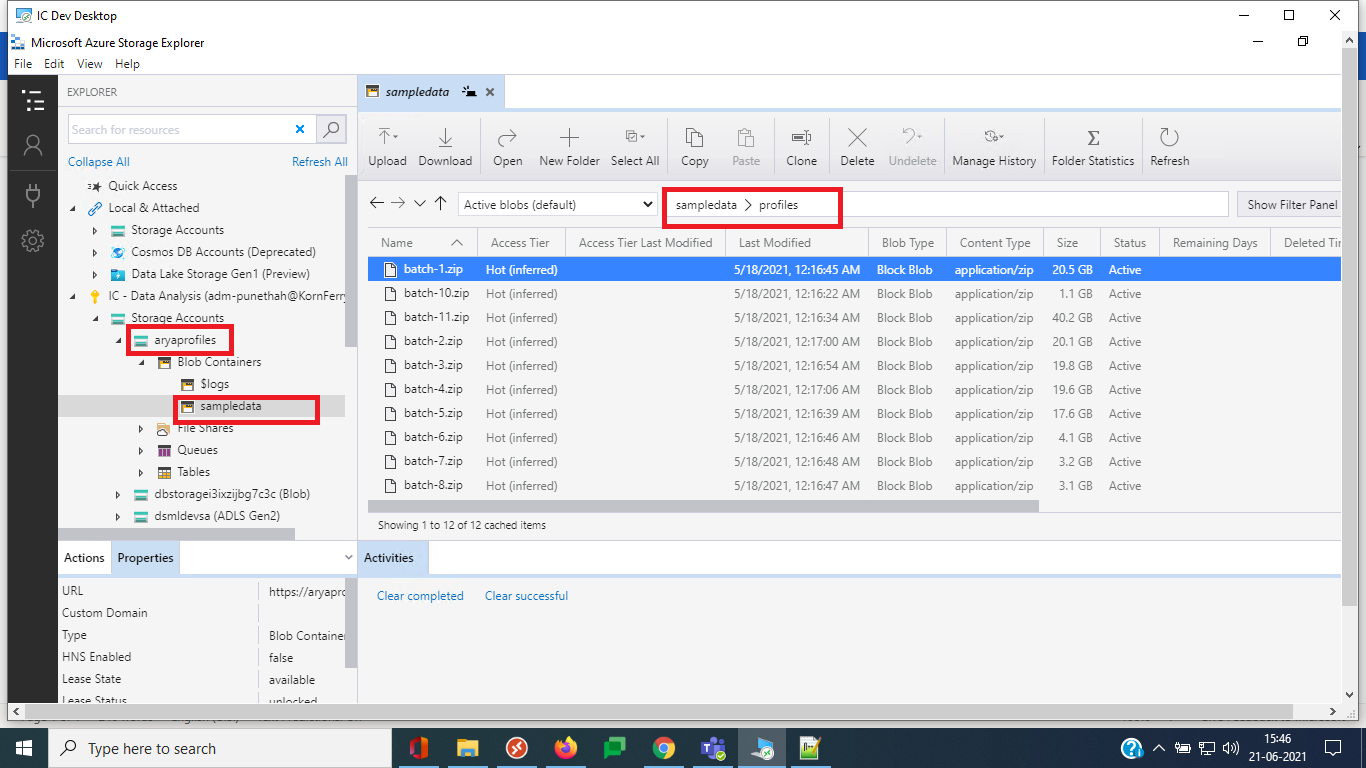
## Command to run:

## Pipeline can be triggered from azure data factory by selecting zipextraction and then clicking Add trigger and then Trigger Now.

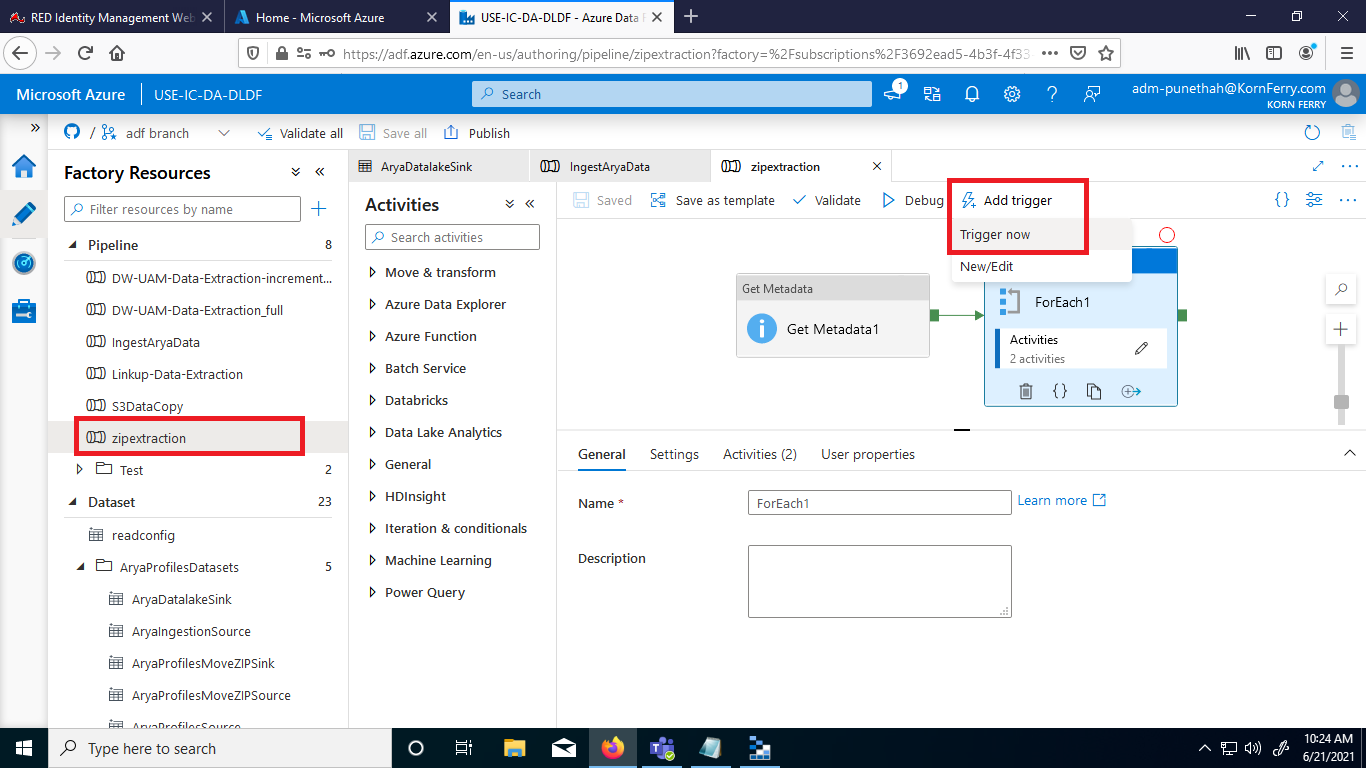


## Steps for Full Load:

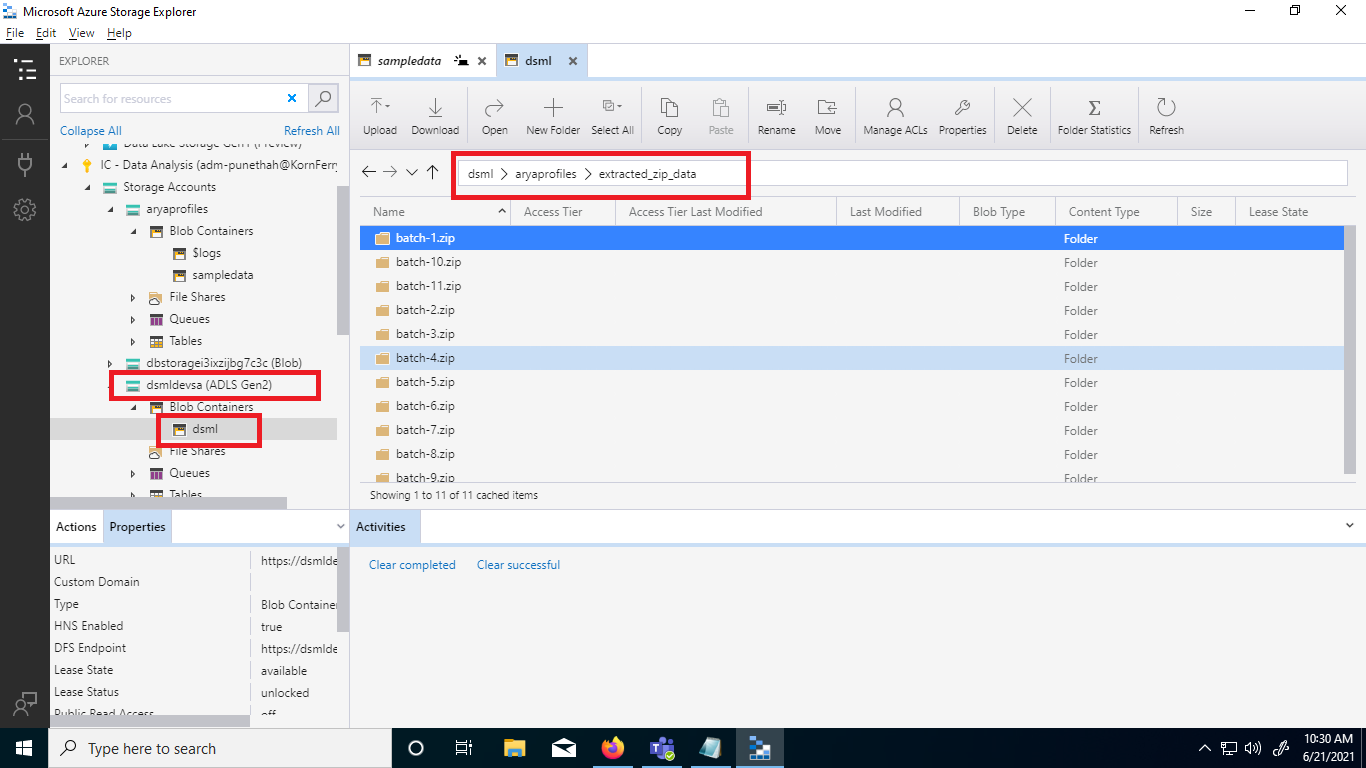
* Copy the new zip files to aryaprofiles storageaccount under sampledata/profiles/ directory as shown below



* Run the zipextraction pipeline from Azure data factory (adf.azure.com )



* Pipeline processes all the zip files in the profiles directory, extracts them to the storage account dsmldevsa in path dsml/aryaprofiles/extracted\_zip\_data  
  The name of the extracted folders ends with .zip (same name as zip files) but they are actually directories containing extracted jsons inside them



* Once a zip file is successfully extracted, the pipeline moves the source zip file present in the storage account aryaprofiles to the path sampledata/adfprocessedzips, so that it doesn't get processed in the future again unless manually copied to source folder again.

## Delta Load (Incremental load)

## JOB NAME: zipextraction

GitHub Link: [Repo Link](https://github.com/HayGroup/mldatalake/tree/databricks/source/notebooks/arya_profile)

## Command to run: Same as in case of Full Load.

## Steps For Delta Load:

Steps are similar to the Full load. In case of Full load, all batches zip files are copied to the source directory and the pipeline runs for all of those zip files and then the zip files are automatically moved to another directory in the same blob (could be deleted if not required). In case of Incremental load, only new zip files are manually copied to source directory.

# Naming & Coding Standards

## Coding Standards

## Table Names

## Data Ingestion Framework Tables

## Data Lake Layer Tables

## Delta Layer Tables: **Profiles in DBFS**

## Variable Names

## Function Names

## Script Names

# Retention Strategy

# Capacity Plan

## Hardware

## Software – Azure Data Factory, Azure Databricks

# Git hub Process

All configuration files related to ADF pipeline are present as json in the repository - <https://github.com/HayGroup/mldatalake/tree/main/adf/>

Git can be connected to ADF using the below configurations as shown below-

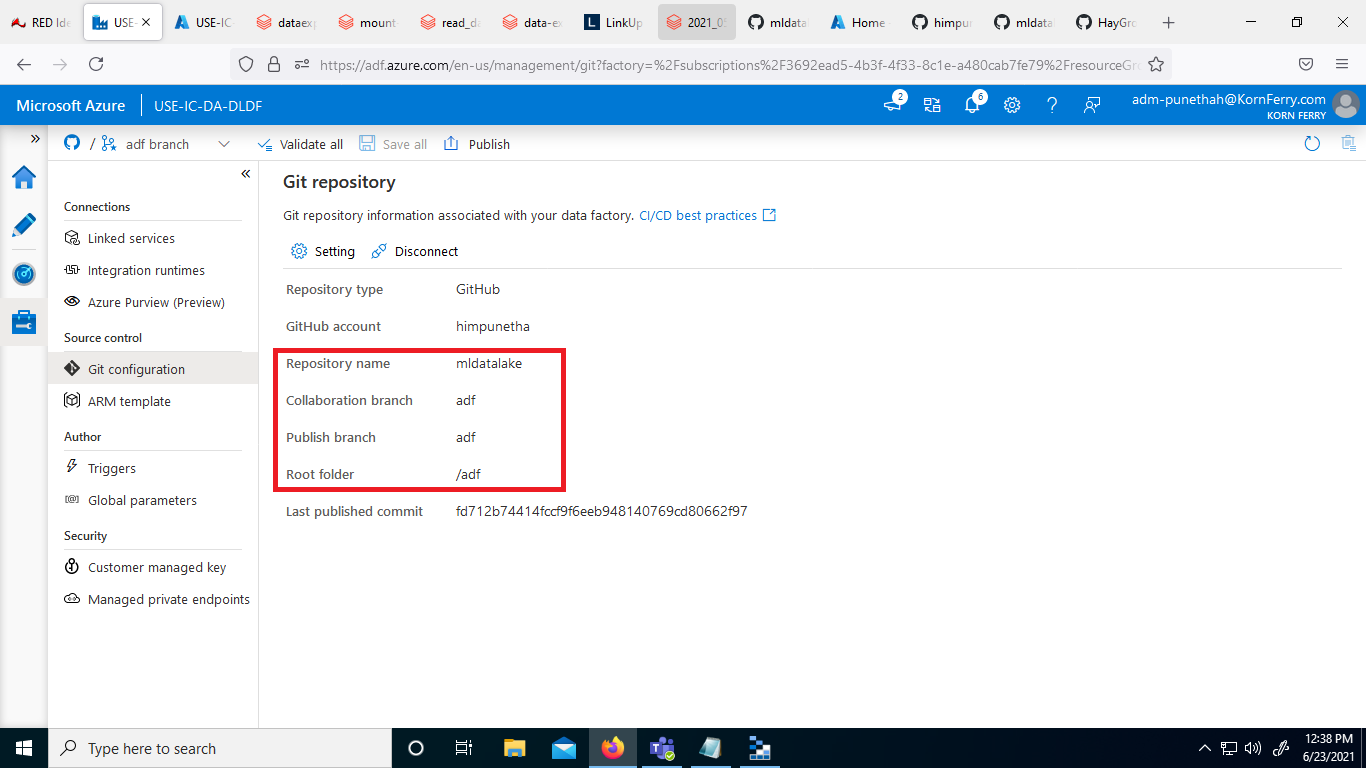
Repository name – mldatalake (<https://github.com/HayGroup/mldatalake/tree/main>)

Collaboration branch – adf

Publish branch – adf

Root Folder - /adf

The new ADF where the code is to be deployed must be created with same name as USE-IC-DA-DLDF . The repository would be needed to be forked in the beginning and then adf can be connected to github with the same configuration as provided above. Select Import existing resources to repository as yes. After connecting, ADF will read the publish\_config.json file present in the adf branch of the repo and then it will import resources. Refresh the adf page once and check if the resources are imported. For more details , check - <https://docs.microsoft.com/en-us/azure/data-factory/source-control>



# Job Orchestration (ADF)

Job Orchestration is done via Azure Data Factory. The details on how to run the pipeline is described in Data Loading Strategy section. The linked services required, Data Sources and activities in ADF pipeline are described in details below.

**ADF - Pipeline Data Sources**

There are 4 data sources defined for ADF pipeline for Aryaprofiles : -

* AryaDatalakeSink - Specifies the destination Data lake gen 2 path - dsml/aryaprofiles/extracted\_zip\_data
* AryaProfilesSource - Specifies the source path but with ZIPDeflate option (for extraction)
* AryaProfilesMoveZIPSource - Specifies source zip folder path path without Zipdeflate option (for moving after extraction is complete)
* AryaProfilesMoveZIPSink - Specifies destination directory path where the source zip file is to be moved once the extraction finishes.

**ADF - Pipeline Activities**

There are total 4 activities in the pipeline

* Get Metadata Activity - to obtain the list of zip files available in the source directory.
* For Each Activity - to run copy operation for all zip files (Operations are performed parallelly).
* Extract (Copy Data Activity)- For extraction of each zip file (present inside For Each).
* Move (Copy Data Activity) - For moving source zip files once extraction finishes successfully (present inside For Each).

# Spark Process

### Requirement

1 - Microsoft Azure Subscriptions

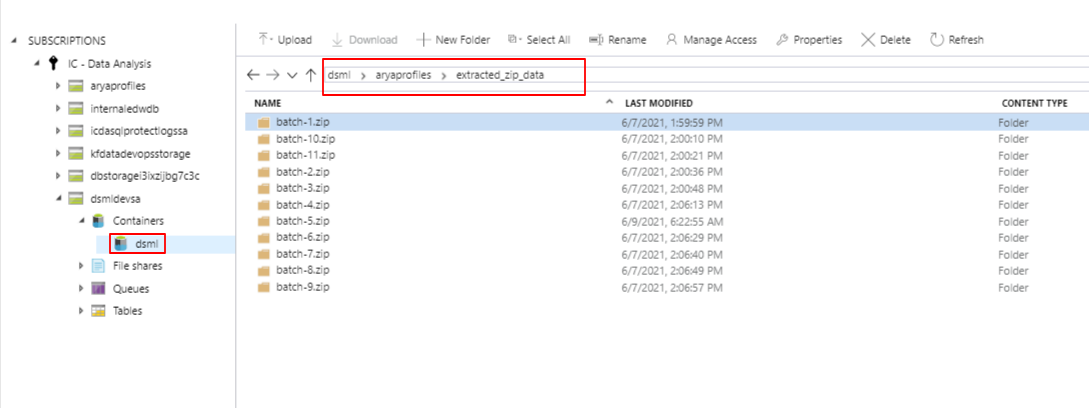
2 - Azure Databricks Instance/Subscription

3 - Databricks Cluster with Scala/SQL/Python language configured.

4 - Notebook that contain multiple commands in sequence for data pipeline to perform ETL

### Process Flow

Data Location where we stored in the ADLS

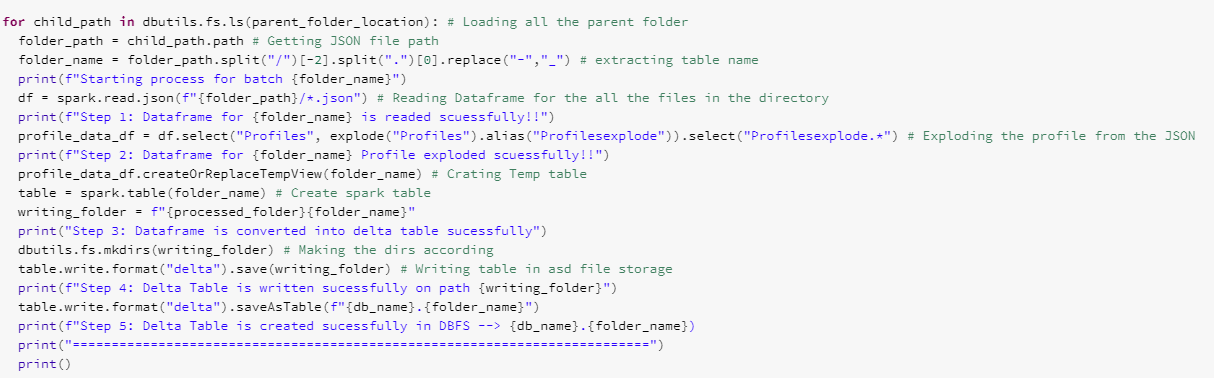


GitHub Link of the [Notebook: https://github.com/HayGroup/mldatalake/blob/databricks/source/notebooks/arya\_profile/read\_data.py](https://github.com/HayGroup/mldatalake/blob/databricks/source/notebooks/arya_profile/read_data.py)

Step 1: Initialize all the required variables in the top of the notebooks so you case use them in the entire notebook



Step 2: Now navigate using dbutils within ADLs GEN2 instance container to the desired files.



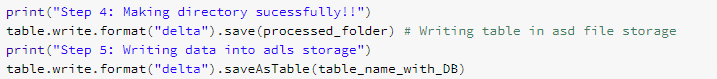
Step 3: Now read that file according to their format in spark data frame.



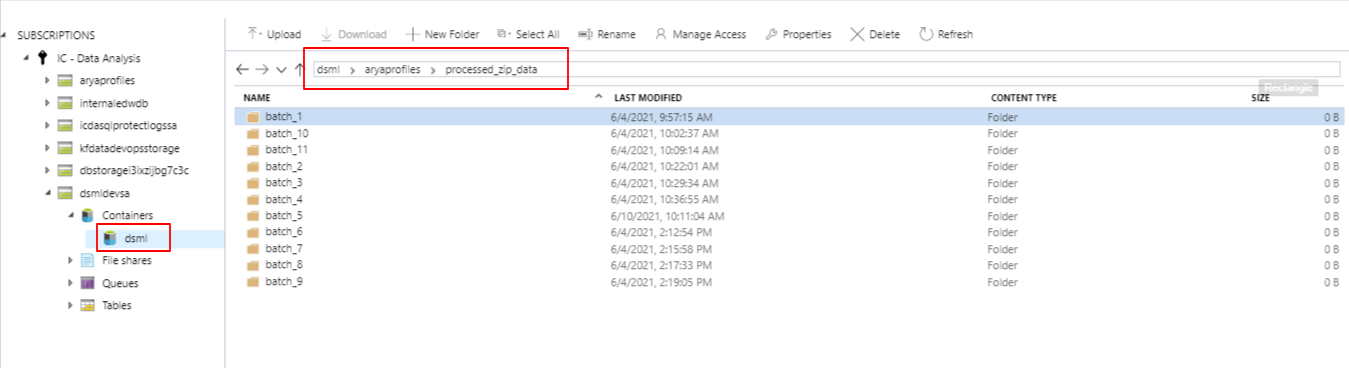
Step 4: Now make temp table in memory form that data frame to check that is compatible to write in delta table.



Step 5 : Once the step 4 is verified now write that temporary table in DBFS in delta format.



Location of ADLS GEN2 where we Dump all the delta tables data.



### Data Used

1 – We can read that data in pyspark in data frame format

2 – We can use that in SQL format also

3 – That Data strictly maintains the ACID properties

4 – We can use in various place for analysis or for model training, The whole fetch on the tip of the finger

# Go Live Process (Dev-Ops)

# Run Book

# Maintenance

# Data Governance