

5. Metadata-driven ADF Framework

Content Owner: [Banga, Nishant \[Non-Kenvue\]](#) and [Bugadi, Shantanu \[Non-Kenvue\]](#)

1. Version History
2. Overview
3. Intended Audience
4. High Level Architecture
5. ELT Process
6. Metadata framework
7. Metadata Model Overview
8. Pre-requisites
9. ADF Pipelines & Configurations
 - a. Execution Overview
 - b. All Master Pipelines
 - c. Child Ingestion Pipelines
 - d. Child Transformation Pipelines
 - e. Child Consumption Pipelines
 - f. Utility pipelines
10. File Validation Overview
11. File Ingestion SP Overview
12. Framework's Modularity and Re-usability
13. Triggers & Schedules
14. Failure Recovery
15. ADF Monitoring, Debugging & Logging - Error and Audit
16. Guide to new developments
17. Unit Testing, SIT in Dev, Any other form of testing before having that confidence to merge into Main
18. Branching
19. Another ADF Instance 002
20. Snowflake Production Deployment using GIT_SQL_Deployment_Utility

S.No	Change History	Modified By	Modified On
1	Initial Draft	Nishant Banga	10 th Sep 2024
2	Second Version	Nishant Banga	23rd Sep 2024

The ADF Framework is built within the Azure Data Factory (ADF) environment, utilizing its powerful capabilities to orchestrate data pipelines. These pipelines are dynamically driven by metadata stored in Snowflake, enabling efficient and coordinated data processing. This Metadata-Driven ADF ELT Platform seamlessly integrates Azure Data Factory with Snowflake's cloud data warehouse to optimize data workflows. Focused on Extract, Load, Transform (ELT) operations, the platform uses a dynamic metadata framework, minimizing the need for extensive coding while allowing for quick adaptation to evolving business requirements.

This documentation is intended for internal team members working on the Metadata-Driven ADF ELT Platform, specifically:

- **Data Engineers** directly involved in implementing and managing the ELT processes.
- **System Architects** responsible for the overall design and integration of the platform within our existing infrastructure.

The High-Level Architecture of the Metadata-Driven ADF ELT Platform integrates key components for optimal data processing and scalability:

- **Azure Data Factory (ADF):** Central to the ELT process, orchestrating data extraction and loading, integrating various data sources, and managing data workflows.
- **Snowflake Cloud Warehousing:** Serves as the data storage and transformation hub, providing scalable, secure, and fast data processing capabilities.
- **Snowflake Metadata Management Layer:** Hosted on Snowflake, this layer manages the metadata driving the ELT processes, allowing dynamic workflow changes based on metadata updates.
- **DBT Integration:** Utilized in the default pipeline configuration for efficient data transformation processes.
- **Source Integration and Connectivity:** Supports all connections facilitated by ADF. Connection specifications are defined partly in Snowflake metadata and in Azure Key Vaults for secure handling of passwords and sensitive details.
- **Consumption Layer Integration:** Within the platform, the consumption-layer Tableau dashboards can be refreshed automatically each time its underlying sources are refreshed. Under consumption, we also have Export jobs and File Copy jobs which is being used by business.

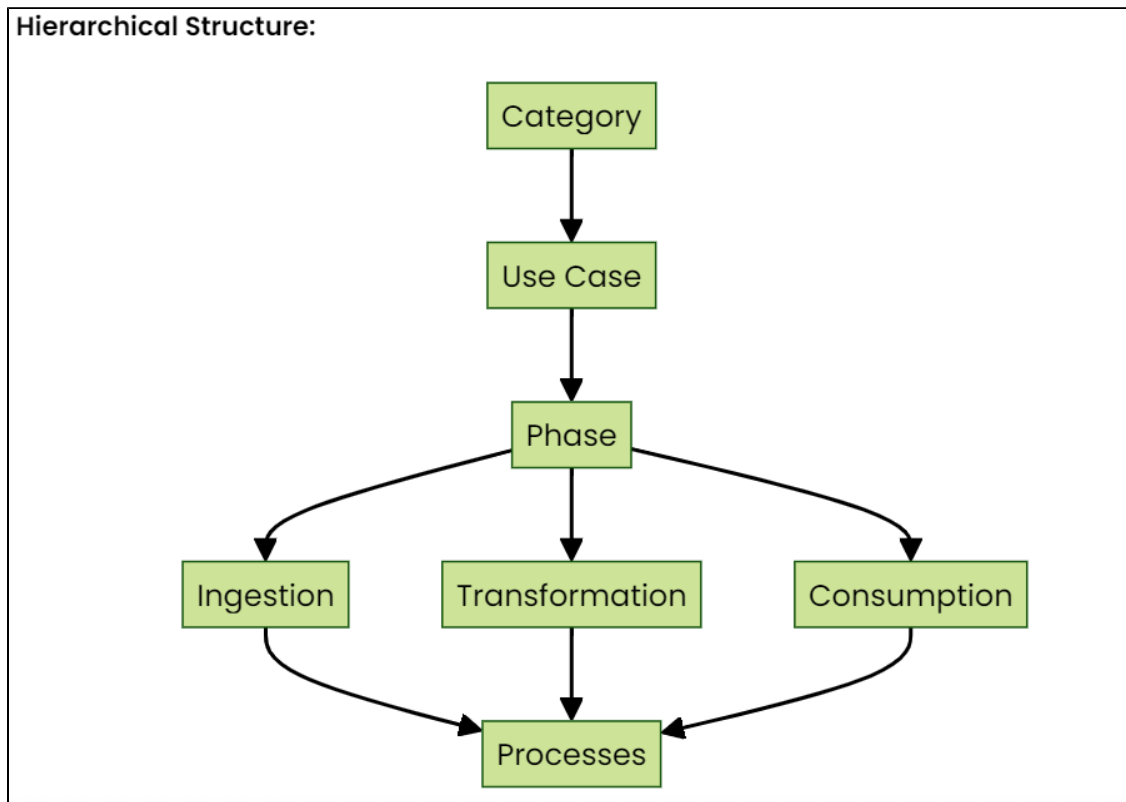
The ELT process in the Metadata-Driven ADF ELT Platform is pivotal for efficient data handling, comprising several key stages:

Metadata Framework

- **Extraction:** Data is extracted from various sources using Azure Data Factory, encompassing a range of formats from different systems. Initially, extracted data is staged in ADLS, providing a scalable and secure environment for raw data.
- **Loading:** The extracted data is loaded from Azure Data Lake Storage (ADLS) to Snowflake. This stage utilizes Snowflake's Copy commands, ADF copy activity and Python SPs to efficiently transfer data. It leverages Snowflake's staging areas and file formats to manage and optimize the data loading process. This integration allows for efficient handling of large volumes of data, maintaining data integrity and consistency.
- **Transformation:** Data is transformed within Snowflake using DBT, involving restructuring, enriching, or aggregating data as per business needs. The transformation logic is metadata-driven, ensuring flexibility and responsiveness to business changes.
- **Consumption:** Equipped with the capability to automatically refresh data visualization tools such as Tableau dashboards post data update, ensuring the latest data insights are always accessible. Also, it extends to extract the data from itg/edw layers after data update and move file from source to target locations.

We have two databases in snowflake - 'DNA_LOAD' & 'DNA_CORE'. DNA_LOAD holds all SDL schemas and SDL tables, along with META_RAW schema for metadata. DNA_CORE holds all rest ITG and EDW schemas and tables.

The Metadata Framework comprises several key entities, each crucial for orchestrating the ELT process:



1. Use-case

- A parent entity to processes, encompassing multiple processes.
- Processes within a usecase are ordered through `sequence_ids`, allowing for sequential or parallel execution patterns.
- Phases are executed in a predefined sequence within each use case.

2. Category:

- The primary driver for pipeline execution, grouping multiple usecases.
- When executing a pipeline, the category is the key parameter passed.
- Each usecase within a category is organized via `sequence_ids`, facilitating orderly execution at a broader level.

3. Phase: Specifies the pipeline phase (e.g., Ingestion, Transformation, Consumption), dictating the order of processes and ensuring an organized workflow.

- **Ingestion:** This phase is responsible for acquiring data from various file sources (Mbox, SFTP, File System) as well as multiple database platforms, including Oracle, AWS, Redshift, RDS, MySQL, and SQL Server.
- **Transformation:** Executes Data Build Tool (DBT) jobs for both ingestion and transformation.
- **Consumption:** Handles the refreshing of Tableau data sources or workbooks, ensuring that the latest data is available for analysis. It also handles export and copy jobs as needed by business.

4. Process:

Metadata Model Overview

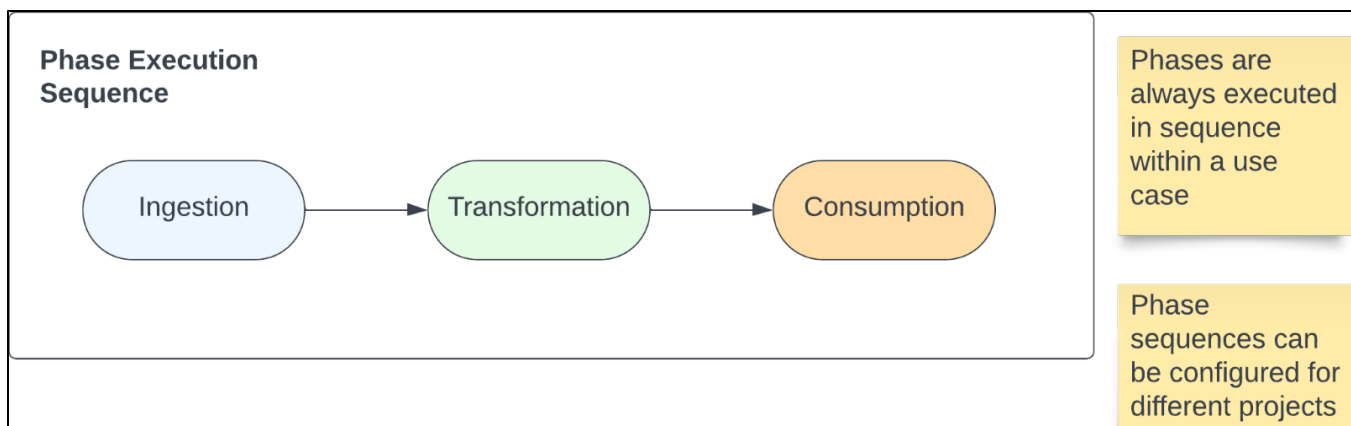
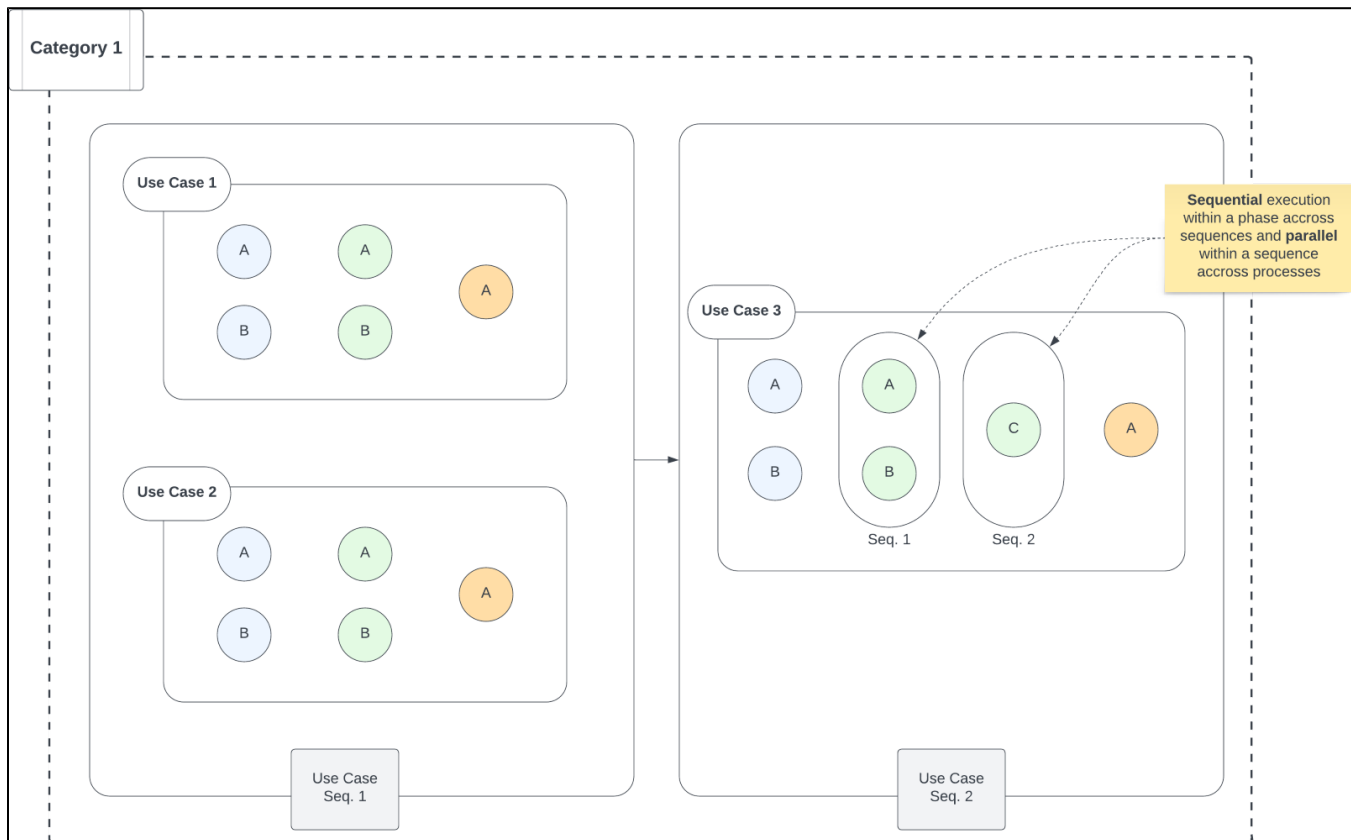
- The core unit within a phase, detailing actions at the table ingestion, dbt model, or dashboard level. Each process is associated with a phase and forms the foundation of the pipeline.
- Processes represent individual tasks within each phase. Examples include file ingestion, Tableau refresh, and DBT job execution.

5. Source:

- Supported sources for data ingestion include Azure Data Lake Storage (ADLS) and Master Data Services (MDS). Additionally, development of an FTP pipeline is planned for future integration.

This structured approach in metadata management ensures precision and flexibility, enabling the platform to effectively handle complex data processing scenarios. The **diagram below** shows an example scenario of a possible pipeline configuration.

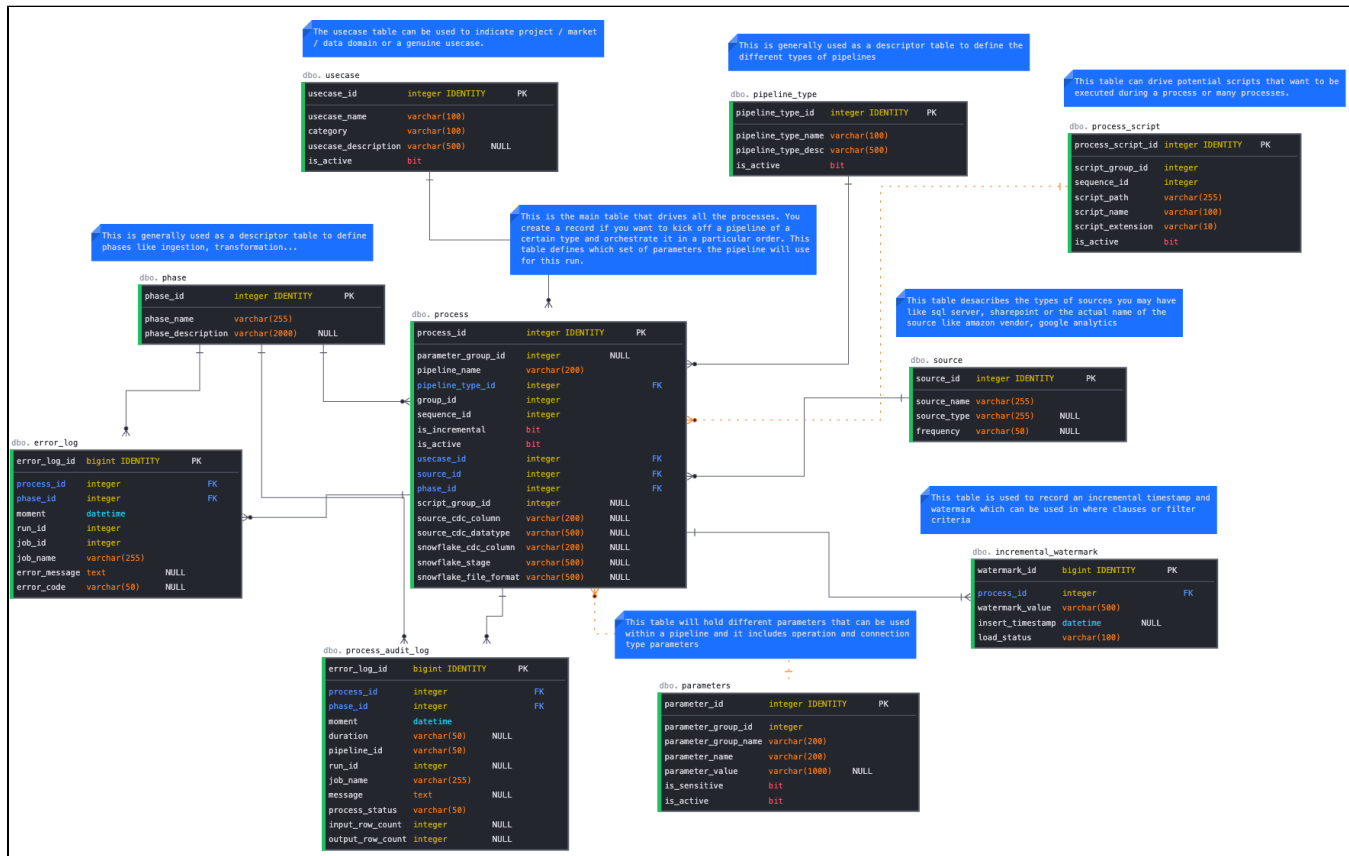
Every set of process (represented by the colored circles) in sequence from left to right. The ones stacked vertically will execute in parallel.



The Metadata Model is crucial to the operation of our ADF ELT Platform, providing a structured framework for managing the ELT process. It comprises a network of tables, each detailing key aspects like data sources, process scripts, and operational phases.

Metadata Model Overview

This section will briefly outline these tables, highlighting their roles in facilitating efficient data management below:



1. Usecase Table:

a. **Purpose:** Manages different use cases, categorizing and sequencing them as per business functions.

b. Key Attributes:

- usecase_id:** A unique identifier for each use case.
- usecase_name:** The name of the use case.
- category:** The category to which the use case belongs. This is the driving pipeline parameter
- usecase_description:** A detailed description of the use case.
- is_active:** Indicates whether the use case is active.
- sequence_id:** Describes the sequence in which usecases will run.

2. Source Table:

a. **Purpose:** Defines the origins of data for the ELT process, identifying and managing data sources.

b. Key Attributes:

- source_id:** Unique identifier for each data source.
- source_name:** Name of the data source.
- source_type:** Type of the data source (e.g., Database, FTP, ADLS, SFTP, File System).
- frequency:** Frequency of data extraction from the source. (Only a descriptive column, no technical significance)

3. Phase Table:

a. **Purpose:** Defines phases in the ELT process, dictating order and nature of each phase.

b. Key Attributes:

- phase_id:** Unique identifier for each phase.
- phase_name:** Name of the phase (e.g., ingestion, transformation, consumption).
- phase_description:** Detailed description of the phase.
- phase_sequence:** Sequence number for phase execution.

4. Parameters Table:

a. **Purpose:** Stores ELT process parameters, crucial for customizing and controlling process flows.

b. Key Attributes:

- parameter_id:** Unique identifier for each parameter.
- parameter_group_id:** Identifier for a group of related parameters.
- parameter_name:** Name of the parameter.
- parameter_value:** The value assigned to the parameter.
- is_sensitive:** Indicates if the parameter holds sensitive information.

Pre-requisites

- vi. **is_active**: Indicates whether the parameter is active.

5. Process Table:

- a. **Purpose**: Central to ELT operations, linking processes with phases, use cases, and sources.
- b. **Key Attributes**:
 - i. **process_id**: Unique identifier for each process.
 - ii. **process_name**: Name of the process.
 - iii. **depends_on**: Signifies if this process depends on any other process id or not.
 - iv. **source_id**: Signifies what is the source for this process. (Relevant only in case of ingestion)
 - v. **is_active**: Indicates whether the process is active.
 - vi. **parameter_id**: Defines the group of parameters for this process in parameters table.
 - vii. **is_incremental**: Signifies if the ingestion process is incremental or not. (Relevant in case of DB ingestion only)
 - viii. **usecase_id, phase_id, group_id, script_id, sequence_id, pipeline_type_id**: Various other attributes linking to phases, use cases, sources, etc., detailing execution sequences and conditions.
 - ix. **snowflake_stage, snowflake_file_format**: Being used to bring these snowflake level parameters for ingestion jobs.
 - x. **phase_type**: Being used by consumption jobs to identify if its a tableau refresh process or export job.

6. Process Audit Log Table:

- a. **Purpose**: Tracks execution logs of processes for monitoring, auditing, and troubleshooting.
- b. **Key Attributes**:
 - i. **audit_log_id**: Unique identifier for each log entry.
 - ii. **process_id, phase_id, moment**: Identifiers and timestamp for process execution.
 - iii. **duration, pipeline_id, run_id, job_name**: Details about the process run.
 - iv. **message, process_status, input_row_count, output_row_count**: Information about process execution and outcome. (Row counts are not being used as of now)

7. Incremental Watermark Table:

- a. **Purpose**: Manages watermarks for incremental loads, ensuring data consistency and effective change tracking.
- b. **Key Attributes**:
 - i. **watermark_id**: Unique identifier for each watermark.
 - ii. **process_id**: Identifier for the related process.
 - iii. **watermark_value**: The value of the watermark.
 - iv. **insert_timestamp**: Timestamp of the watermark insertion.
 - v. **load_status**: Status of the data load.

8. Error Log Table:

- a. **Purpose**: Logs errors during ELT processes, essential for error tracking and maintaining data integrity.
- b. **Key Attributes**:
 - i. **error_log_id**: Unique identifier for each error log entry.
 - ii. **process_id, phase_id, moment**: Identifiers and timestamp of the error occurrence.
 - iii. **run_id, job_id, job_name**: Details about the job during which the error occurred.
 - iv. **error_message, error_code, category**: Information about the nature and type of error.

9. Category_Market_Map:

- a. **Purpose**: Maps the usecases' categories with the market. It is being used by Tableau monitoring of ADF pipelines
- b. **Key Attributes**:
 - i. **category**: Distinct categories from usecase tables
 - ii. **market**: Name of the market

10. Process Script Table: (Currently Not in Use, but can be used in future developments)

- a. **Purpose**: Manages scripts for operational aspects of data processing, detailing sequence and specifics.
- b. **Key Attributes**:
 - i. **process_script_id**: Unique identifier for each script.
 - ii. **script_group_id**: Identifier for the group of related scripts.
 - iii. **sequence_id**: Sequence number for script execution.
 - iv. **script_path**: Path where the script is located.
 - v. **script_name**: Name of the script file.
 - vi. **script_extension**: File extension of the script.
 - vii. **is_active**: Indicates whether the script is active for use.

11. Pipeline Type Table: (Currently Not in Use, but can be used in future developments)

- a. **Purpose**: Categorizes pipeline types, aiding in workflow differentiation and management.
- b. **Key Attributes**:
 - i. **pipeline_type_id**: Unique identifier for each pipeline type.
 - ii. **pipeline_type_name**: Name of the pipeline type.
 - iii. **pipeline_type_desc**: Description of the pipeline type.
 - iv. **is_active**: Indicates whether the pipeline type is active.

Workflow:

- **Triggers within ADF initiate the master pipeline corresponding to a specific category**, thereby initiating the execution of associated processes.
- The **master pipeline serves as the entry point to the framework**, orchestrating the execution of processes within the designated category.

ADF Pipelines & Configuration

- **Not all tableau datasources/workbooks are triggered from ADF.** Most of these are live connections in tableau and gets refreshed as soon as data is refreshed. If for a particular datasource/workbook, you don't see an active consumption process in Metadata, it will be a Live connection.

Usage:

1. Category Definition:

- Categories are defined within the metadata to group related processes together.
- Triggers within ADF are configured to activate master pipelines associated with specific categories.

2. Master Pipeline:

- Acts as the central control point for executing processes within a category.
- Triggered by ADF based on predefined conditions, initiating the execution of associated processes.

The ADF framework operates as a metadata-driven system, comprising master and child pipelines tailored to various functionalities within data processing workflows.

Execution Overview

- The execution initiates when a trigger activates the "master" pipeline, using a category specified in the use case table stored in Snowflake.
- Subsequently, the "master" pipeline triggers the "usecase master" pipeline to determine the execution order and requirements of use cases, whether sequential or parallel.
- The "usecase master" pipeline further triggers the "phase master" pipeline to identify and order the phases of execution, also in sequential or parallel fashion.
- Based on the phase ID:
 - If it's 1, the "phase master" pipeline calls the "ingestion master" pipeline.
 - If it's 2, it calls the "transformation master" pipeline.
 - If it's 3, it calls the "consumption master" pipeline.
- When all phases are to be executed, they follow the sequence of ingestion, transformation, and consumption.
- Hence the ingestion master, transformation master and consumption master pipelines are kicked off based on the configuration.

All master Pipelines

master pipeline

This serves as the entry point of the framework. It begins by verifying whether another instance is already running for the same category. If an instance is detected, the execution is halted, and an error is generated. Otherwise, the process proceeds. Next, it checks for any dependencies, ensuring that all prerequisite jobs have been completed. Based on the results, the execution will either pause or terminate with an error. Afterward, the framework retrieves all distinct use case sequences and executes the "usecase master" for each, either sequentially or in parallel, depending on the configuration.

Parameter	Default Values	Any comments
category	-	Mandatory parameter for category name
force_rerun	false	Can be used to rerun the instance even if older one was under execution or stopped in the middle
ignore_deps	false	Can be used to bypass the ignore dependency feature.

usecase master pipeline

This pipeline is triggered by the 'Master' pipeline for a specific use case sequence ID. It retrieves all use cases within the current sequence and initiates the 'Phase Master' pipeline.

Parameter	Default Values	Any comments
category	-	Mandatory parameter for category name
usecase_sequence_id	-	sequence id for the usecase/s being triggered

phase master pipeline

This pipeline extracts and processes information for all phases of a given use case, executing in the sequence of ingestion, transformation, and consumption. If a specific phase is not required, it is automatically skipped. In the event of ingestion failure, the transformation phase will still proceed. However, if the transformation phase fails, the consumption phase will not be executed.

Parameter	Default Values	Any comments

ADF Pipelines & Configuration

category	-	Mandatory parameter for category name
usecase_id	-	usecase id for the usecase being triggered

ingestion master pipeline

This pipeline identifies the unique sequence IDs required for the ingestion phase of a specific use case. Once the distinct sequence IDs are determined, it initiates the 'Ingestion Sequence' pipeline.

Parameter	Default Values	Any comments
category	-	Mandatory parameter for category name
usecase_id	-	usecase id for the usecase being triggered
phase_id	-	Value should be 1 in case of ingestion

ingestion sequence pipeline

This pipeline retrieves all processes associated with a specific sequence ID from the process table for ingestion under same category. It processes these in parallel, in batches of 20. Depending on the source system for each process, the execution flow is directed to the corresponding pipeline. For instance, the ADLS pipeline is triggered when the source is ADLS, and the SFTP pipeline is invoked when the source is SFTP. Additional details regarding these pipelines can be found in the "Other Pipelines" section.

Parameter	Default Values	Any comments
category	-	Mandatory parameter for category name
usecase_id	-	usecase id for the usecase being triggered
phase_id	-	Value should be 1 in case of ingestion
sequence_id	-	Sequence id of the processes to be triggered

transformation master pipeline

This pipeline identifies the unique sequence IDs required for the transformation phase of a specific use case. Once the distinct sequence IDs are determined, it initiates the 'Transformation Sequence' pipeline.

Parameter	Default Values	Any comments
category	-	Mandatory parameter for category name
usecase_id	-	usecase id for the usecase being triggered
phase_id	-	Value should be 2 in case of transformation

transformation sequence pipeline

This pipeline retrieves all processes associated with a specific sequence ID from the process table for transformation under same category. It processes these in parallel, in batches of 10. The execution flow is directed to the "DBT" pipeline.

Parameter	Default Values	Any comments
category	-	Mandatory parameter for category name
usecase_id	-	usecase id for the usecase being triggered
phase_id	-	Value should be 2 in case of transformation
sequence_id	-	Sequence id of the processes to be triggered

consumption master pipeline

This pipeline identifies the unique sequence IDs required for the consumption phase of a specific use case. Once the distinct sequence IDs are determined, it initiates the 'Consumption Sequence' pipeline.

Parameter	Default Values	Any comments

ADF Pipelines & Configuration

category	-	Mandatory parameter for category name
usecase_id	-	usecase id for the usecase being triggered
phase_id	-	Value should be 3 in case of consumption

consumption sequence pipeline

This pipeline retrieves all processes associated with a specific sequence ID from the process table for consumption under same category. It processes these in parallel, in batches of 10. Depending on the 'phase_type' for each process, the execution flow is directed to the corresponding pipeline. For instance, the tableau pipeline is triggered when the phase_type is tableau, and the Data Export pipeline is invoked when the phase_type is Data_export. Additional details regarding these pipelines can be found in the "Other Pipelines" section.

Parameter	Default Values	Any comments
category	-	Mandatory parameter for category name
usecase_id	-	usecase id for the usecase being triggered
phase_id	-	Value should be 3 in case of consumption
sequence_id	-	Sequence id of the processes to be triggered

Child Ingestion Pipelines

DB pipeline

The database (DB) pipeline is triggered when the source ID is 5. This pipeline is designed for use with database servers such as SQL, MySQL, Oracle, Redshift, RDS, Salesforce, and others. It extracts parameters from a designated parameter table and reconstructs the SQL query for incremental data loads. The pipeline can either be utilized solely for data extraction into Azure Data Lake Storage (ADLS) or for loading data into the SDL tables via direct copy or stored procedure (SP). If the parameters include a truncate load flag, the pipeline will truncate the target table accordingly. Additionally, the pipeline invokes a child pipeline, "DB_To_ADLS," which performs the data extraction. This pipeline is typically used in conjunction with the ADLS pipeline, facilitating a two-step process: unloading files from the source and loading them into Snowflake.

Parameter	Default Values	Any comments
category	-	Mandatory parameter for category name
usecase_id	-	usecase id for the usecase being triggered
phase_id	-	Value should be 3 in case of consumption
sequence_id	-	Sequence id of the processes to be triggered
process_id	-	Process id of the process to be triggered

Reverse SQL pipeline

It is triggered when the source ID is 9. This pipeline is responsible for reverse-syncing tables from Snowflake to MDS. It utilizes a source query, provided through metadata parameters, which is executed against Snowflake. The resulting data is unloaded into ADLS and then loaded into the MDS tables. The pipeline allows for selecting the source system (either Snowflake Load or Snowflake Core).

The 'truncate_and_load' flag determines whether the target table will be truncated before loading the data. Additionally, the 'truncate_source' flag controls whether the source Snowflake tables should be truncated after the sync is complete.

Data synchronization occurs only when the source query returns data.

Parameter	Default Values	Any comments
category	-	Mandatory parameter for category name
usecase_id	-	usecase id for the usecase being triggered
phase_id	-	Value should be 3 in case of consumption
sequence_id	-	Sequence id of the processes to be triggered
process_id	-	Process id of the process to be triggered

SFTP pipeline

It is triggered when the source ID is 4. This pipeline is designed to transfer source files from various SFTP servers to Azure Data Lake Storage (ADLS) for further processing. The pipeline's routing logic is determined by the 'ftpName' parameter in the metadata, allowing it to manage different SFTP servers accordingly. Initially, the pipeline checks for the existence of files that need to be transferred. It then invokes the child pipeline, 'SFTP_To_ADLS,' which moves the files from the SFTP server to ADLS.

There are specific exceptions based on the SFTP server in terms of file retrieval. For example, for the POP master server, only files from the previous day are retrieved, while for the Iqvia server in the Pacific region, only the latest file is transferred. Additionally, different wildcard expressions are used to search for files on some servers.

For more detailed information about handling files for a specific server, please refer to the ADF code for that server. This pipeline is typically used in conjunction with the ADLS pipeline, facilitating a two-step process: unloading files from the source and loading them into Snowflake.

Parameter	Default Values	Any comments
category	-	Mandatory parameter for category name
usecase_id	-	usecase id for the usecase being triggered
phase_id	-	Value should be 3 in case of consumption
sequence_id	-	Sequence id of the processes to be triggered
process_id	-	Process id of the process to be triggered

File System pipeline

This pipeline is triggered when the source ID is set to 7. Its primary function is to transfer source files from various file system servers to Azure Data Lake Storage (ADLS). The pipeline determines the appropriate route for the source server based on the 'FILESYSTEMNAME' parameter specified in the metadata.

A special handling case applies to the server 'sawsbtagpw0000', where over 1,000 files are received daily. In this instance, the files are compressed into a zip file before being transferred to ADLS. The zipped file is then used for loading data into SP.

This pipeline is typically used in conjunction with the ADLS pipeline, facilitating a two-step process: unloading files from the source and loading them into Snowflake.

Parameter	Default Values	Any comments
category	-	Mandatory parameter for category name
usecase_id	-	usecase id for the usecase being triggered
phase_id	-	Value should be 3 in case of consumption
sequence_id	-	Sequence id of the processes to be triggered
process_id	-	Process id of the process to be triggered

MDS pipeline [DECOMMISSIONED]

This pipeline is triggered when the source ID is set to 2. It was previously utilized to execute CDATA sync jobs via REST APIs, transferring data from MDS systems to Snowflake. However, it has been decommissioned following the migration to the DB pipeline.

Parameter	Default Values	Any comments
category	-	Mandatory parameter for category name
usecase_id	-	usecase id for the usecase being triggered
phase_id	-	Value should be 3 in case of consumption
sequence_id	-	Sequence id of the processes to be triggered
process_id	-	Process id of the process to be triggered

MDS Reverse Sync pipeline [DECOMMISSIONED]

This pipeline is triggered when the source ID is set to 6. It was previously used to execute CDATA synchronization tasks via REST APIs. Its primary function was to reverse-sync tables from Snowflake to MDS. However, it has been decommissioned following the migration to the Reverse SQL pipeline.

ADF Pipelines & Configuration

Parameter	Default Values	Any comments
category	-	Mandatory parameter for category name
usecase_id	-	usecase id for the usecase being triggered
phase_id	-	Value should be 3 in case of consumption
sequence_id	-	Sequence id of the processes to be triggered
process_id	-	Process id of the process to be triggered

ADLS pipeline

This pipeline is triggered when the source ID is set to 1 and is responsible for loading data from files stored in ADLS. These files may originate directly from mbox, be generated by a SQL pipeline, or be unloaded from SFTP or File Server pipelines.

The process begins by fetching files based on the file prefix provided in the metadata parameters. If the 'is_truncate' flag is set, the SDL table is truncated, but this only occurs when there are files available for processing. For cases where truncation is required regardless of file availability, the 'is_forced_truncate' parameter is used.

Each file is processed sequentially. First, the file extension is validated. Upon successful validation, the 'load_file' pipeline is invoked for each file, which performs additional validation and loads the data.

Parameter	Default Values	Any comments
category	-	Mandatory parameter for category name
usecase_id	-	usecase id for the usecase being triggered
phase_id	-	Value should be 3 in case of consumption
sequence_id	-	Sequence id of the processes to be triggered
process_id	-	Process id of the process to be triggered

Child Transformation Pipelines

DBT Pipeline

This pipeline is initiated whenever a DBT job needs to be triggered. Initially, metadata parameters are extracted, followed by retrieving the necessary credentials for the DBT API from Azure Vault. Using the account_id and job_id parameters, a DBT job is triggered via REST API. The process then monitors the status of the triggered job at intervals defined by the 'wait_time' parameter to determine whether it has completed or is still in progress. Once the job is marked as complete, the execution transitions to the 'DBT_utility_check' pipeline.

Parameter	Default Values	Any comments
category	-	Mandatory parameter for category name
usecase_id	-	usecase id for the usecase being triggered
phase_id	-	Value should be 3 in case of consumption
sequence_id	-	Sequence id of the processes to be triggered
process_id	-	Process id of the process to be triggered

DBT Check Utility Pipeline

This pipeline is triggered upon the completion of a DBT job. There are two types of DBT jobs: ingestion and transformation.

- **For transformation jobs:** If the status response of the DBT job is 10, it indicates a successful execution, and the process is terminated.
- **For ingestion jobs:** An additional Data Quality (DQ) check is performed. Based on the temp_id parameter, relevant models are identified. A stored procedure is then called using the temp_id to pinpoint any failed models. For each failed model, the corresponding DQ check results are retrieved and communicated to business users via the designated channel. In cases where the DBT job itself fails, this is detected at the beginning of the pipeline. For each failed model, a notification is sent, and an error is logged. However, for DQ check failures, a single audit log entry is created with the message 'FAIL: DQ_CHECK' to represent the overall failure.

Child Consumption Pipelines

Tableau Pipeline

This pipeline is triggered by the 'consumption_sequence' pipeline when the `phase_type` is set to 'tableau'. It is responsible for initiating the refresh of a Tableau datasource or workbook. The type of refresh—whether datasource or workbook—is determined by the 'refreshType' parameter in the metadata. Although the process is identical for both cases, the URLs used for the API calls differ slightly.

Once metadata parameters are retrieved, the pipeline fetches the UUID and password required to authenticate the Tableau API. Using these credentials, a sign-in request is executed, generating a token for subsequent API calls. If the refresh is for a datasource, the datasource ID is retrieved based on its name; for a workbook, the workbook ID is retrieved using the workbook name. This ID is then used to initiate the Tableau refresh.

Given that Tableau refreshes can be time-consuming, the status is recorded in the audit log as 'Refresh started separately.' A separate child pipeline, 'tableau_refresh_status,' is then triggered to monitor the status of the refresh.

Parameter	Default Values	Any comments
category	-	Mandatory parameter for category name
usecase_id	-	usecase id for the usecase being triggered
phase_id	-	Value should be 3 in case of consumption
sequence_id	-	Sequence id of the processes to be triggered
process_id	-	Process id of the process to be triggered

Tableau Refresh Status Pipeline

This pipeline is triggered by the 'tableau' pipeline when a refresh is initiated. It continuously monitors the status of the refresh job at intervals specified by the 'wait_time' parameter to check whether the job has completed or is still running. Once the job is finished, its response is evaluated and categorized as either a success or failure. Based on this outcome, appropriate error and audit logs are generated. Afterward, the process signs out from the Tableau server using the API token. The success or failure of this pipeline determines whether the Tableau refresh was successful.

DCL_ContactHist_Unload_MBox

This pipeline is triggered by the 'consumption_sequence' pipeline when the `phase_type` is set to 'DCL_ContactHist_Unload_MBox'. Its primary function is to export the necessary data and load the file to an SFTP location utilized by the business. It accepts a parameter, 'excluded_columns,' which specifies the columns to be excluded from the export for the specified table. Using the information schema, the pipeline constructs a query with these parameters to retrieve unique 'chno' values from the table. For each distinct 'chno', a file is generated and placed in the ADLS folder. Once all files are created, they are zipped into a folder named with the current date, and the zipped file is then transferred to the SFTP location.

Parameter	Default Values	Any comments
category	-	Mandatory parameter for category name
usecase_id	-	usecase id for the usecase being triggered
phase_id	-	Value should be 3 in case of consumption
sequence_id	-	Sequence id of the processes to be triggered
process_id	-	Process id of the process to be triggered

Data Export Pipeline

This pipeline is triggered by the 'consumption_sequence' pipeline when the `phase_type` is set to 'Data_Export'. Its primary responsibility is to export the required data and load the file into ADLS. The pipeline uses the 'unloading_query' parameter to extract data from the specified database source (e.g., DNA CORE). Whether the generated file includes headers is determined by the 'headerNeeded' parameter, and whether the data is enclosed in quotes is controlled by the 'quotesEnclosed' parameter. This pipeline is designed to work in conjunction with the 'Mbox_File_Copy' pipeline in sequence.

Parameter	Default Values	Any comments
category	-	Mandatory parameter for category name
usecase_id	-	usecase id for the usecase being triggered

ADF Pipelines & Configuration

phase_id	-	Value should be 3 in case of consumption
sequence_id	-	Sequence id of the processes to be triggered
process_id	-	Process id of the process to be triggered

MBox File Copy Pipeline

This pipeline is triggered by the 'consumption_sequence' pipeline when the 'phase_type' is set to 'Mbox_File_Copy'. Its main function is to move the exported file from the previous step. Depending on the 'prefix_flag' or 'suffix_flag', the current date is appended to the target file name. The 'encoding' parameter is used to convert the exported file to the specified target encoding. If the 'split_file' parameter is enabled, the file is split into multiple files, each containing a maximum of 2,000 records. Finally, the processed file is stored in the target folder path specified in the parameters, where it will be available for business use.

Parameter	Default Values	Any comments
category	-	Mandatory parameter for category name
usecase_id	-	usecase id for the usecase being triggered
phase_id	-	Value should be 3 in case of consumption
sequence_id	-	Sequence id of the processes to be triggered
process_id	-	Process id of the process to be triggered

Utility Pipelines

Load File Pipeline

This pipeline is responsible for loading files into Snowflake. After the file is picked by the ADLS pipeline, it is passed to this pipeline for further assessment, transformation, validation, and data loading.

Upon receiving the necessary parameters, the pipeline first determines whether the source file is an Excel document and if conversion to CSV is required. If conversion is necessary, the Excel file is transformed into CSV format. In cases where the Excel file contains multiple sheets, the pipeline generates separate CSV files for each sheet.

Next, based on the validation flag parameter, the file undergoes validation for file name, extension, and headers. Once the validation is successfully completed, the process moves forward to load the data. The appropriate data loading method is selected based on the 'load_method' parameter, with the following options:

- **copy**: Directly copies the CSV file using the COPY command. This method is used when no additional ETL columns are required.
- **sp**: Executes a stored procedure for complex data loading logic.
- **JapanDCL**: A special case for Japan DCL files, where data is directly loaded, and required ETL columns are added post-load.
- **Japan**: A special case for Japan POS files, where ETL columns are added after a direct load. This route includes additional logic to handle specific timestamp values.
- **PROX**: A special case for PROX files under EOM Regional, where data is directly loaded and ETL columns are added during the loading process.
- **otcSellout**: A special case for OTC Sellout files in the China market, where data is directly loaded and required ETL columns are added post-load.
- **RG_Clavis**: A special case for RG_Clavis files under EOM Regional, where direct loading is performed with ETL columns added during the process.

Once the data has been successfully loaded, the source files are archived and moved to the "Processed" folder. The process concludes with an audit log entry.

File Archival Pipeline

This pipeline is triggered by the 'Load_File' pipeline to archive files in ADLS. It archives files by moving them to the processed/archive folder. In case of an error, files are moved to the processed/error folder. Additionally, the pipeline is responsible for deleting any temporary intermediary files.

DB to ADLS Pipeline

This pipeline is triggered by the 'DB' pipeline to extract data from a specified database server and unload it into ADLS. It supports multiple existing database connections, which can be reviewed and configured through the pipeline and its parameters.

SFTP to ADLS Pipeline

This pipeline is triggered by the 'SFTP' pipeline to transfer files from a specified SFTP server and location into ADLS, with the option to archive the file on the SFTP server (archiving logic is currently commented out but can be enabled after JnJ is decommissioned). It supports

File Validation Overview

multiple existing SFTP connections, which can be reviewed within the pipeline and its parameters. In cases where the source file is zipped and the unzipped version is required in ADLS, the pipeline unzips the file, loads it into ADLS, and deletes the original zip file uploaded to ADLS.

Write Audit Log Pipeline

The Audit Log pipeline is responsible for logging and tracking the execution of processes within the framework. This pipeline ensures comprehensive traceability and accountability throughout the data processing workflow. The audit log captures Start and Finish/ Fail for the pipelines which duration of pipeline run.

Write Error Log Pipeline

The Error Log pipeline is designed to capture and manage errors encountered during pipeline execution. It records error messages, timestamps, and other relevant information to facilitate troubleshooting. By centralizing error logging, this pipeline enables swift identification and resolution of issues to minimize disruptions.

NotifyTeamsChannel Pipeline

This pipeline facilitates the delivery of notification messages to both support and business channels. We categorize notifications into two types:

1. **Support Notifications:** These contain detailed information about specific failures, along with a URL directing users to the ADF monitoring tab for the relevant run.
2. **Business Notifications:** These are sent to respective channels based on market (e.g., ASP for the Asia-Pacific region, CHN for China, IND for India, etc.), focusing on errors relevant for business teams to take corrective actions.

All notification channels are organized under the 'AP DataOps' team. The 'Core Support' channel is designated for support notifications, while other channels cater to business notifications.

Key parameters within the pipeline include:

- **container:** Configures the webhook for business channels.
- **isSupportNotification:** Indicates whether the notification is for support.
- **isBusinessNotification:** Indicates whether the notification is for business.
- **isError:** Specifies if the notification pertains to an error.
- **supportMessage:** Defines the error message content for the support notification.
- **businessMessage:** Defines the error message content for the business notification.

The pipeline leverages several conditional components to distinguish between support and business notification logic. A special case exists for the Pacific market, where notifications are triggered upon successful Tableau datasource refresh. This is an exception, as the primary focus of notifications remains on error reporting.

File Structure Validation performs **three operations**:

- File Name Check
- File Extension Check
- File Header Check

File header validation is triggered only upon the successful completion of both the file name and file extension checks.

Validation Parameter: 1-1-1

1 = Active

0 = Not Active

The three digits in the validation parameter indicate the following:

The first occurrence represents file name validation.

The second occurrence represents file extension validation.

The third occurrence represents file header validation.

File Name Validation

Each market has a dedicated function. Upon receiving a file, the corresponding function is called for preprocessing. After preprocessing, the file name validation function is executed to verify the file name.

Parameters Required:

val_file_name: This parameter is used for validating the name of the incoming file.

index: Defines the portion of the filename to retain or remove (Options: Pre, Full, Last, First). It helps remove dates or other characters from the file name:

pre: Removes the prefix from the filename (e.g., 202408_filename.csv becomes filename.csv).

full: Retains both the filename and the date, if present (e.g., filenameA_082024 remains unchanged).

last: Removes the suffix from the filename (e.g., filenameA_082024.csv becomes filenameA).

first: Removes the prefix from the filename (e.g., I_fileA_20240813.csv becomes fileA).

File Extension Validation

The file extension validation function is called to verify the file's extension. Once both file name and file extension validations are successful, it checks if file header validation is required.

Parameters Required:

val_file_extn: This parameter validates the file extension (e.g., csv, XLSX, txt).

File Header Validation

In file header validation, there are three main steps:

- a. Extraction Step: The file is read, and the header is extracted.
- b. Preprocessing Step: The header names from both the file and the parameters are cleansed and formatted into a specific structure (i.e., a list of strings) for validation.
- c. Header Checks: The header validation function is called to validate the headers. It performs the following checks and returns specific messages:
 - i) Check if the count of header names from the file matches the parameter values.
 - ii) Check if the file header contains any extra columns.
 - iii) Check for unmatched columns.

Parameters Required:

folder_path: The location of the file.

file_header_row_num: The row number where the header is present in the file.

val_file_header: The expected header names to validate against the file's actual header.

sheet_names (if required): If header checks are required for more than one sheet in an Excel file.

file name: The file name to read and process the file.

If a validation failure occurs at any stage, the File Structure Stored Procedure (SP) returns a specific error message. If all checks pass, the File Structure Validation SP returns a "Success" message.

This section outlines the approach for reading files from Azure Data Lake Storage (ADLS), transforming the data using Pandas and Snowpark, and loading it into Snowflake tables. The procedure is designed to preprocess various file formats such as CSV, XLSX (single and multiple sheets), ZIP files, and more, which are uploaded to a Snowflake stage. The processed data is then loaded into a target table (SDL table) after structural transformations like filtering and adding custom columns (e.g., file name, current date). Additionally, the transformed output is saved in a success folder.

This stored procedure enables the seamless ingestion and preprocessing input data files from a Snowflake stage. The process involves unzipping, filtering, transforming, and loading the data into a Snowflake table based on requirement, while also saving the processed data to a success folder. The robust error handling and transformation logic ensure that any issues encountered during execution are caught and reported effectively.

Input Parameters

Param (ARRAY): This array contains the following elements:

- Param[0]: file_name - Name of the file to be processed.
- Param[1]: stage_name - The stage where the file is stored.
- Param[2]: temp_stage_path - Path within the stage.
- Param[3]: target_table - Name of the table where the data will be loaded.

Logic Overview:

1. **Session Setup**
 - a. The procedure starts by setting up the Snowflake session and extracts the database, schema, and target table details from the parameters.
2. **DataFrame Schema Definition**
 - a. Defines a schema using snowflake.snowpark.types.StructType to read the input file into a Snowflake DataFrame, based on the file's structure and requirements.
3. **Reading Data**
 - a. Main Data: Loads the input file into a Snowflake DataFrame, skipping rows as needed.
 - b. Header Data: Reads the file headers separately for location names, especially when processing
 - c. For Excel files, pandas.read_excel() is used to read the data into a Pandas DataFrame.
4. **Extracting Unwanted Columns**
 - a. Using the select() function, only the necessary columns for ingestion into the SDL table are selected, ignoring unwanted columns.
5. **Filtering and Conversion**
 - a. Row-level filters are applied based on specific criteria.
 - b. Converts the remaining data into a Pandas DataFrame if additional transformations are required.
6. **Transformation Logic:** The procedure performs various transformations such as:
 - a. Pivoting
 - b. Filtering
 - c. Adding custom columns like:
 - i. FILE_NAME: The name of the processed file.
 - ii. CRT_DTTM: The current timestamp (Asia/Singapore time zone).
 - iii. RUN_ID: A unique identifier for the process run as per JnJ.

Framework's Modularity and Re-usability

- d. Structural changes as per the business requirement.
 - e. Data Cleaning: Functions such as trim(), upper(), and regexp_replace() are applied to clean the data.
 - f. Date Management: Dates are handled using current_timestamp and to_date() functions with timezone conversion managed by the pytz library.
 - g. Null Value Management: Rows with all null values are dropped.
 - h. Concatenation: Functions like concat() and format_number() are used to format the data for readability.
 - i. If no data remains after processing, the procedure returns "No Data in file."
7. **Final DataFrame**
 - a. The transformed data is compiled into a new DataFrame that contains the required columns from the input file.
 8. **Writing Data to Target Table**
 - a. Before loading new data, the target SDL table is truncated since all SDL tables follow a "truncate and load" process.
 - b. The final DataFrame is written into the target table in append mode.
 - c. If the DataFrame is empty, the procedure returns "No Data in file."
 9. **Saving Processed Data to Success Folder**
 - a. The processed data is saved as a CSV file to the success folder in the specified stage.
 10. **Handling ZIP Files**
 - a. If the input file is a ZIP archive, the procedure extracts and reads all contained CSV files, concatenating them into a single Pandas DataFrame for further processing. This has been applied, for example, in the HCP market processing (e.g., "1_TerritoryMaster_202406.zip").

Exception Handling:

- **KeyError:** Handles missing columns in the DataFrame and provides a clear error message.
- **MergeError:** Catches DataFrame merging errors.
- **General Exceptions:** Any other errors are caught, and the relevant details are logged for troubleshooting.

The framework is designed to be loosely coupled and highly functional, with a modular structure that enhances usability and maintainability. By mastering the fundamentals of orchestrating use cases and understanding the phases and sequencing, you can seamlessly integrate complex functionalities in a step-by-step manner. For any new development, it is advisable to deconstruct the problem and leverage existing pipelines and components before initiating new development efforts.

Trigger Names
cd_scorecard_health_inventory_table_refresh_trigger
CHN_OTC_SELLOUT_trigger
coufang_ppm_file_trigger
datamart_refresh_customer_COGS_trigger
DM_INTEGRATION_DLY_REFRESH_sunday_trigger
EOM_ECOMM_rg_mrkt_RG_CLAVIS_trigger
EOM_OKR_trigger
EOM_PROX_trigger
ETL_XDM_TLRSR_LOAD_TRIGGER
GCH_trigger
Global_OTIF_EDL_DNA_trigger
GT_SELLOUT_OTC_TRIGGER
GT_SELLOUT_trigger
HCP360_IN_Ventasys_Load_trigger
HCP_OneSea_trigger
hk_daily_sales_trigger
ID_IVY_trigger
ID_POS_DAILY_trigger
ID_POS_trigger
ID_PS_trigger
IDN_MDS_SDL_trigger
IDN_MDS_trigger_1
IDN_MDS_trigger_2

Triggers & Schedules

IDN_MDS_trigger_3
IDN_MDS_trigger_4
IDN_MDS_trigger_5
IDN_MDS_trigger_6
in_hcp360_iqvia_kpi_datamart_trigger
In_Finance_Simulator_Ingestion_trigger
IN_GA360_trigger
IN_GEO_TAGG_DATAMART_trigger
IN_HCP360_IQVIA_Ingestion
IN_HCP360_MDS_Data_Refresh_trigger
IN_MDS_REFRESH_PERFECT_STORE_TRANS_trigger
IN_MI_DASHBOARD_DATAMART
IN_MI_MSL_PERF_DATAMART_trigger
IN_MI_PSR_DATAMART_trigger
IN_MT_SELLIN_VS_SELLOUT_DATAMART_trigger
IN_SFMC_LOAD_trigger
IN_SKU_RECOMMENDATION_trigger
IN_SKURECOM_GT_SSS_2024_DATAMART_trigger
IN_SKURECOM_MSLSPIKE_DATAMART_trigger
IN_SSS_Scorecard_Trigger
IN_XDM_Ingestion_trigger
India_RT_trigger
India_Salesman_Target_trigger
INDIA_ETL_XDM_SDLITG_LOAD_trigger
INDIA_KEY_ACCOUNT_OFFTAKE_trigger
INDIA_SDLITG_PWC_KAS_TBL_trigger
j_ecommerce_nts_mds_load_trigger
j_india_generic_source_excel_sdl_data_ingestion_trigger
j_kr_pos_prc_condition_trigger
j_kr_price_tracker_trigger
j_kr_tw_sales_target_trigger
j_tw_sell_in_forecast_transaction_data_ingestion_trigger
J_AP_IDC_ETL_FRAMEWORK_trigger
J_OS_DNA_SDL_TO_EDW_WM_SALES_trigger
J_PH_DNA_TAB_REFRESH_trigger
J_RG_WATSONS_INV_MY_PH_TW_trigger
japan_pos_mds_sync_trigger
JP_DCL_AFFILIATE_CANCEL_LOAD_trigger
JP_DCL_CINEXT_trigger

Triggers & Schedules

JP_DCL_GENERIC_EXTRACTION_FILESETUP_trigger
JP_DCL_MDS_trigger
JP_DCL_ML_PREDICTION_LOAD_trigger
JP_DCL_MYKOKYA_trigger
JP_DCL_RAKUTEN_trigger
JP_DCL_SFMC_DCL_ITG_trigger
JP_MDS_Reverse_trigger
JP_MDS_trigger
JP_Sellout_MDS_trigger
JPN_DCL_SFTP_trigger
JPN_EDI_MASTER_INVENTORY_WORKFLOW_trigger
JPN_EDI_MASTER_WEEKEND_trigger
JPN_POS_LOAD_trigger
JPN_SELLIN_INTEGRATION_trigger
JPN_TSP_MASTER_trigger
JS_EX_PRODUCT_ATTR_DIM_trigger
kr_pos_emark_trigger
KR_DADS_trigger
KR_ECOMMERCE_1_trigger
KR_MDS_COUPANG_PREMIUM_trigger
KR_onpack_target_trigger
KR_OTC_ECOM_INV_FRAMEWORK_trigger
KR_SFMC_NAVER_DATA_trigger
KR_TRADE_PROMOTION_trigger
m_edw_mysales_trigger
MALAYSIA_POS_trigger
market_mirror_trigger
my_gt_sales_dna_to_mds_trigger
my_list_price_trigger
my_sellout_inv_sales_trigger
MY_JOINT_MONTHLY_trigger
MY_PERFECT_STORE_trigger
MY_POS_SIPOS_trigger
MY_SELLIN_ANALYSIS_trigger
MY_SELLOUT_SELLIN_MDS_trigger
MY_SELLOUT_SISO_morning_evening_trigger
NA_MDS_KR_POS_trigger_1
NA_MDS_KR_POS_trigger_2
NA_MDS_TW_POS_IMS_trigger
NA_TW_IMS_trigger

Triggers & Schedules

open_order_detail_extract_trigger
OTIF_D_Trigger
PCF_ANZ_PERENSO_DISTRIBUTION_trigger
PCF_ANZ_PERENSO_WEEKLY_LOAD_trigger
PCF_CBG_DATA_INGESION_trigger
PCF_Cogs_ETL_Framework_trigger
PCF_Inventory_Health_trigger
PCF_IRI_SCAN_SALES_trigger
PCF_MDS_trigger
PCF_PA_PHARMA_trigger
PCF_PAC_PHARM_trigger
PCF_PERENSO_METCASH_trigger
PCF_PERENSO_trigger
PCF_PHARM_ECOMM_trigger
PCF_REFRESH_trigger
PCF_ROI_REFRESH_trigger
PCF_TRAX_trigger
PCF_Weekly_Forecast_trigger
PH_ACOMMERCE_DATA_INGESTION_trigger
PH_ANALYSIS_SISO_NPI_SCORECARD_trigger
PH_BP_TARGET_IOP_TARGET_trigger
PH_MDS_POS_PERFECTSTORE_DMS_trigger
PH_NON_POS_trigger
PH_SFMC_CRM_trigger
PH_TAB_REFRESH_trigger
POP6_MASTER_TRANSACTION_trigger
POP6_refresh_Trigger
RE_Summary_Details_Execution_trigger
Regional_Ecomm_One_View_trigger
Regional_Price_Tracker_trigger
REGIONAL_COPA_trigger
rg_travel_retail_refresh_trigger
Sellin_Integration_Job_saturday_trigger
sg_scan_refresh_trigger
sg_sellout_refresh_trigger
TH_CRM_SFMC_trigger
TH_CUSTOMER360_trigger
TH_FILE_INGESTION_trigger
TH_MT_Daily_price_load_trigger
TH_PERFECTSTORE_trigger

Failure Recovery

TH_Tesco_Transdata_trigger
THAILAND_trigger
TW_IMS_DISTRIBUTOR_trigger
TW_SFMC_CRM_trigger
TW_STRATEGIC_CUSTOMER_HIERARCHY_trigger
TWSI_TARGET_INGESTION_trigger
UAT_J_RG_SELL_IN_COPA_TAB_REFRESH_trigger
UAT_pka_tableau_refresh_trigger
UAT_rg_mds_to_dna_refresh_trigger
UAT_SAP_BW_MASTER_AND_ACCT_ATTR_trigger
UAT_SAP_BW_trigger
VN_DMS_trigger
VN_MT_PERFECTSTORE_trigger
VN_MT_trigger
VN_TARGET_TOPDOOR_trigger
Yimian_Price_Ingestion_CHN_MDS_trigger

There are multiple approaches to failure recovery, depending on the type, point, and impact of the failure. Below are some examples:

- 1) **ADF Server Issues:** If a pipeline is stopped or fails due to an Azure Data Factory (ADF) server issue, ADF monitoring can be used to locate the pipeline and rerun it using the 'Retry from Failure' option.
- 2) **Snowflake Server Issues:** In case of Snowflake server problems, the pipeline may or may not run since the first step involves logging and querying metadata. Similar to ADF issues, you can use ADF monitoring to retry the pipeline from the failure point.
- 3) **Manual Pipeline Triggering Options:**
 - **Re-run all use cases in a category:** Safely re-run the master pipeline for the specific category.
 - **Re-run use cases in a category with sequence:** Re-run the use case master pipeline for the category and specific sequence ID.
 - **Re-run a specific use case within a category:** Re-run the phase master pipeline for the category and use case ID.
 - **Re-run a specific phase for a use case:** Depending on the phase (ingestion, transformation, or consumption), re-run the respective master pipeline - ingestion master pipeline, transformation master pipeline, consumption master pipeline.
 - **Process-only execution:** If you need to run only specific processes, trigger the appropriate pipeline directly with the required parameters. The correct pipeline will automatically call subsequent child pipelines as designed.
 - **Exclude certain processes from an entire job:** Mark the unnecessary processes as inactive in the production metadata, then trigger the pipeline at the category, use case, or process level. Remember, you can set use cases, processes, and parameters to inactive if they are not needed in the framework.

To start, there are two primary metadata tables responsible for logging: the Process Audit Log and the Error Log.

The **Audit Log** table captures execution logs for monitoring, auditing, and troubleshooting of processes.

The **Error Log** table records any errors encountered during ELT processes, which is critical for error tracking and ensuring data integrity.

1) Steps to debug the pipeline failures using ADF

We can get the details of pipeline runs through the tableau dashboards ([ADF reporting](#)), which provides us the details on total number of pipelines that ran successfully and number of pipelines that failed. To get more details (or) to deep dive into granular level, we use the audit tables in the snowflake to check the pipeline runs. It gives more flexibility for the users to filter based on category / market etc, if they use the snowflake audit tables.

The first level of analysis to check on how a pipeline ran (or) to check for any failures, it will be through the Process Audit log table in meta_raw schema.

1.1) To check the overall run stats for all the pipelines.

We can use the process_audit_log table for this.

ADF Monitoring, Debugging & Logging - Error and Audit

The **moment** column in the above table holds the timestamp details at which a specific pipeline started its execution. This column can be used to filter the pipeline stats for any specific day.

The **process_status** column can be used to check the status of the pipelines. There are 3 statuses (START, FINISH, FAIL, REFRESH_STARTED_SEPARATELY, FAIL:DQ_CHECK)

- START – If the pipeline has started its execution
- FINISH – If the pipeline has completed its execution
- FAIL – If a pipeline has failed during execution
- REFRESH_STARTED_SEPARATELY - Marks the end of tableau pipeline that trigger has been initiated, but refresh is still ongoing.
- FAIL:DQ_CHECK - Marks the failure of DBT job in case some DQ checks fail.

Note: The timestamp details in the moment column are based on SGT time zone.

```
Select * from process_audit_log where moment >= '2024-04-15 00:00:00.000' order by audit_log_id desc;

Select * from process_audit_log a join category_market_mapping c on a.category=c.category where a.moment >= '2024-05-15 00:00:00.000' order by c.market, a.category, a.audit_log_id asc;
```

1.2) To check the pipelines that has failed on a specific date

Like mentioned above the error_log table can be used to check the pipelines that has failed in execution.

The **moment** column in the error_log table can be used to check the pipelines that has failed for any specific day.

The **ERROR_MESSAGE** column provides the complete description on why a specific pipeline got failed.

```
Select * from error_log where moment >= '2024-04-18 00:00:00.000' order by error_log_id desc;
Select * from error_log e join category_market_mapping c on e.category=c.category where e.moment >= '2024-04-15 00:00:00.000' order by c.market, e.category, e.error_log_id asc;
```

1.3) To check the overall pipeline run stats based on market level granularity

To check the overall pipeline run stats based on market level, we should join the process_audit_log table with category_mapping table.

We can use the below query and change the market to whichever market that we need these details for.

```
Select * from process_audit_log a join category_market_mapping c on a.category=c.category where a.moment >= '2024-05-15 00:00:00.000' and c.market='PH_SFMC_CRM' order by c.market, a.category, a.audit_log_id asc;
```

To check the pipeline run stats based on market and category level granularity. The below provided query can be used.

```
Select * from process_audit_log a join category_market_mapping c on a.category=c.category where a.moment >= '2024-04-15 00:00:00.000' and c.market='REGIONAL' and a.category='REGIONAL_MDS' order by c.market, a.category, a.audit_log_id asc;
```

1.4) To check the failed pipelines based on market level granularity

To check the pipelines that failed on market level, we should join the error_log table with category_mapping table.

To check the failed pipeline run stats for a specific market, we can use the below query and change the market to whichever market that we need these details for

```
Select * from error_log e join category_market_mapping c on e.category=c.category where e.moment >= '2024-05-15 00:00:00.000' and c.market='PH_SFMC_CRM' order by c.market, e.category, e.error_log_id asc;
```

To check the failed pipelines of a specific market and category. The below provided query can be used.

```
Select * from error_log e join category_market_mapping c on e.category=c.category where e.moment >= '2024-04-15 00:00:00.000' and c.market='REGIONAL' and e.category='REGIONAL_MDS' order by c.market, e.category, e.error_log_id asc;
```

1.5) To identify a data source that has failed during a tableau refresh failure.

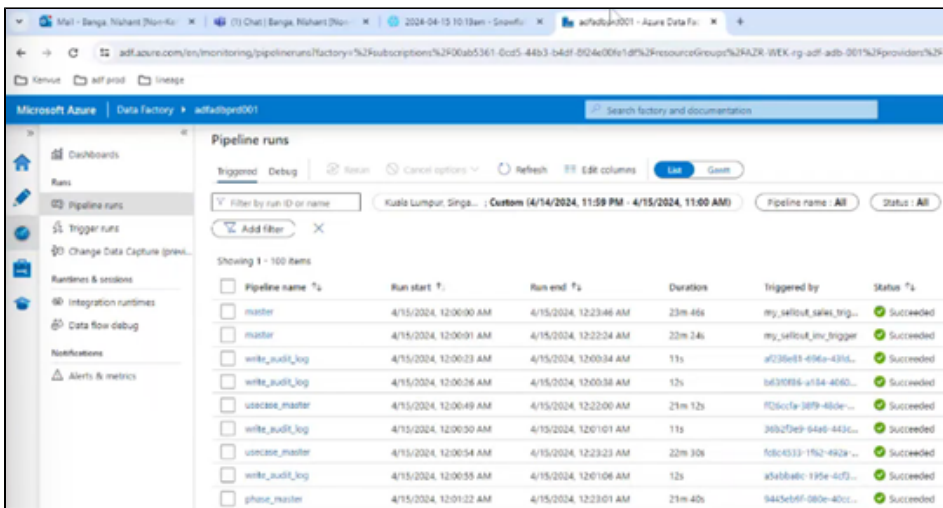
Guide to new developments

To identify the data source that has failed due to a tableau refresh failure, we need to join the process_audit_log / error_log table with the parameter table to identify the corresponding data source that has failed or succeeded. This can be filtered using process_id and phase_id from process_audit_log / error_log tables. The below provided query can be used to check all the tableau data sources and their associated process_id. In a similar way, we can also check for DBT failure as well.

```
Select process.usecase_id,process.process_id, process.process_name, process.parameter_group_id, parameters.parameter_value tableauSourceName from process join parameters on process.parameter_group_id= parameters.parameter_group_id where parameters.parameter_name='tableauSourceName' and tableauSourceName in ('SM and Pure Gold - Selfservice','PH SFMC Membership Analysis','PH NPI','PH GT Scorecard DS');
```

1.6) To further investigate on any failures

we can directly check the details in the ADF. Once we login to ADF, we need to switch to monitor section and choose pipeline runs option if it's not selected already, as shown in the below Figure .

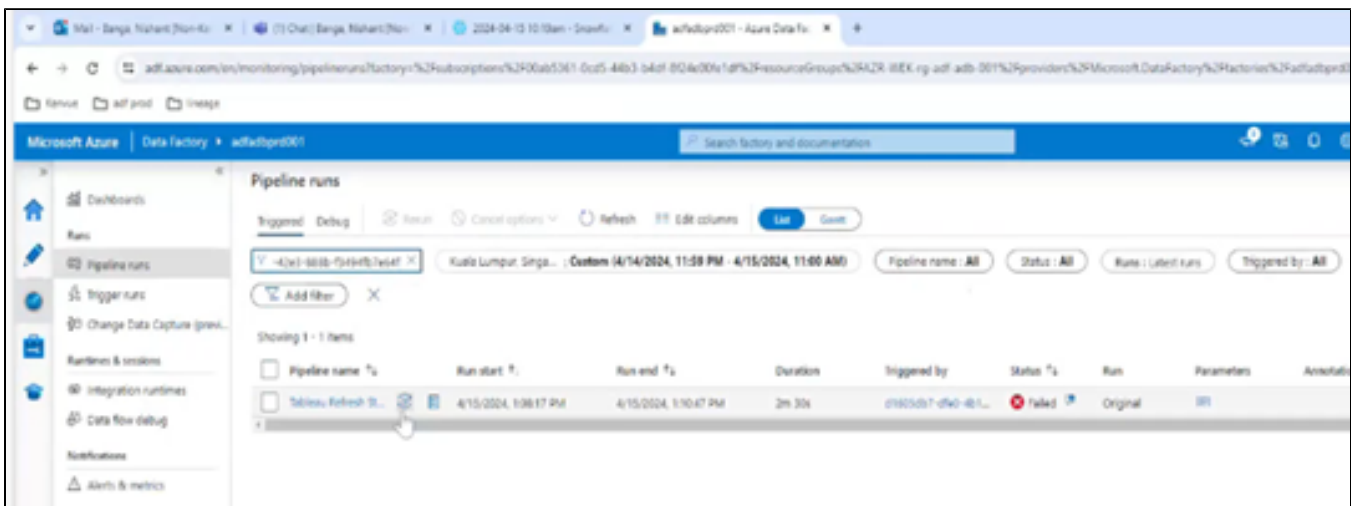


Pipeline name	Run start	Run end	Duration	Triggered by	Status
master	4/15/2024, 12:00:00 AM	4/15/2024, 12:23:46 AM	23m 46s	my_sailout_sales_trig...	Succeeded
master	4/15/2024, 12:00:01 AM	4/15/2024, 12:22:34 AM	22m 34s	my_sailout_inv_trigger	Succeeded
write_audit_log	4/15/2024, 12:00:23 AM	4/15/2024, 12:00:34 AM	11s	a0238e81-496a-436a...	Succeeded
write_audit_log	4/15/2024, 12:00:36 AM	4/15/2024, 12:00:38 AM	12s	b6330986-a154-4060...	Succeeded
usecase_master	4/15/2024, 12:00:49 AM	4/15/2024, 12:22:00 AM	21m 12s	f026c1fa-3879-460e...	Succeeded
write_audit_log	4/15/2024, 12:00:50 AM	4/15/2024, 12:01:01 AM	11s	36b270ef-9449-443c...	Succeeded
usecase_master	4/15/2024, 12:00:54 AM	4/15/2024, 12:23:23 AM	22m 30s	f0bc4533-1f62-492a...	Succeeded
write_audit_log	4/15/2024, 12:00:55 AM	4/15/2024, 12:01:06 AM	12s	a54b4abc-195e-4c72...	Succeeded
ghour_master	4/15/2024, 12:01:22 AM	4/15/2024, 12:23:01 AM	21m 40s	944c60ff-080e-401c...	Succeeded

The displayed window provides the high-level details of the pipeline runs like Pipeline name, run start time, run end time, duration it took to complete the run, triggered by details, status of the pipeline (succeeded, in progress, queued, failed, cancelled) and Run Id.

We can use the multiple filter options that are available to view only the specific pipeline runs.

To check the details of a specific run from the process_audit_log / error_log tables, we can copy the run_id from the snowflake tables and search in the ADF to get the complete pipeline and run detail of that specific run_id as shown in Figure.



Pipeline name	Run start	Run end	Duration	Triggered by	Status	Run	Parameters	Actionable
Tableau.Refresh St...	4/15/2024, 1:38:17 PM	4/15/2024, 1:10:47 PM	2m 30s	a7625d5b7-ef40-481c...	Failed	Original

Note: We don't have to check the ADF web app often. This check is required only when there are multiple failures happening repeatedly and we are not able to figure out the reason with the error message displayed from the error_log table.

For error monitoring using the MS Teams notification channel. Please refer to above section for 'NotifyTeamsChannel' pipeline section.

Here is the guide to new developments:

1. **Tableau Data Source/Workbook Refresh via ADF**
 - a. Create a new process with a phase ID of 3.
 - b. Follow the procedures and parameters for any process with a phaseType 'tableau'.
 - c. Ensure this process is added under an existing use case or a new one, based on the requirements.
2. **DBT Job Execution via ADF**
 - a. Create a new process with a phase ID of 2.
 - b. Follow the procedures and parameters for any process with phase ID 2.
 - c. Identify whether the new DBT job involves ingestion or transformation. For ingestion, ensure that the DQ metadata is inserted into Snowflake, as the ADF DQ checks rely on this metadata.
3. **Mbox File Ingestion**
 - a. Create a new process with source ID 1.
 - b. Follow the procedures and parameters for any process with phase ID 1 and source ID 1.
 - c. If the file is not a direct load, develop the required stored procedure for the data load, and ensure that the loadType parameter is set correctly.
4. **SFTP File Ingestion**
 - a. Create two new processes with source IDs 4 and 1, in sequence.
 - b. Follow the procedures and parameters for processes with phase ID 1 and source IDs 4 and 1.
 - c. For the first process, ensure that the SFTP route is present in the ADF flow. If the route or connection doesn't exist, update the pipeline to accommodate the new route.
 - d. For the second process, develop the required stored procedure if the file is not a direct load, and ensure the loadType parameter is correctly specified.
5. **File System File Ingestion**
 - a. Create two new processes with source IDs 7 and 1, in sequence.
 - b. Follow the procedures and parameters for processes with phase ID 1 and source IDs 7 and 1.
 - c. For the first process, ensure the file server route is present in the ADF flow. If the route or connection is missing, update the pipeline to include this new route.
 - d. For the second process, develop the required stored procedure if the file is not a direct load, and ensure the loadType parameter is correctly specified.
6. **Database (DB) Ingestion**
 - a. Create two new processes with source IDs 5 and 1, in sequence.
 - b. Follow the procedures and parameters for processes with phase ID 1 and source IDs 5 and 1.
 - c. For the first process, ensure the database server route is included in the ADF flow. If the connection or route does not exist, enhance the pipeline accordingly.
 - d. For the second process, develop the required stored procedure if the file is not a direct load, and ensure the loadType parameter is set correctly.
 - e. Alternatively, you may create a single process with source ID 5 that performs the data load. In this case, ensure the load_method is specified instead of "none."
7. **MDS Ingestion**
 - a. Create a new MDS process with source ID 5.
 - b. Follow the procedures and parameters for existing MDS processes.
8. **Reverse MDS Sync**
 - a. Create a new reverse sync process with source ID 9.
 - b. Follow the procedures and parameters for existing reverse MDS sync processes.

Unit testing is always performed in the developer's own branch during feature development. Unit testing is considered complete only when the pipeline is tested with the correct metadata from Snowflake, and no hardcoded settings are used.

After merging the developer's branch into the release branch, it is essential to conduct regression testing on any related pipelines or processes to ensure that the new development does not impact existing functionality. A round of testing in the release branch is highly recommended.

For ingestion processes, it's advisable to test all possible permutations and combinations during development. Developers are encouraged to introduce manual errors to intentionally fail pipelines for negative testing, but must resolve these errors once testing is complete.

Developers should avoid keeping their changes in their branch for extended periods, as ADF can present challenges with merge conflicts. We've observed issues where incorrect connections form during conflict resolution when working directly with JSON. Therefore, it's best to make changes, test them, and promptly merge them into the release branch. If needed, reapply changes by pulling the latest code from the release branch or main.

Ensure you check audit and error logs during your test runs.

After unit testing, proceed with integration testing. This involves running the pipeline not just on its own, but from various master pipelines within the framework, to ensure no critical metadata is missed.

For file ingestion stored procedures, always run them in Snowflake first before executing the process directly through ADF.

Our branch management strategy is as follows: for every release, a release branch is created from the main branch. Developers create their own branches from this release branch, perform their development and testing, and then merge their changes back into the release branch.

Subsequently, a pull request (PR) is raised to merge the release branch into the main branch, which triggers production deployment upon approval and merge. It's important to note that when a PR is raised from the release branch to the main branch, a development deployment occurs. Production deployment only happens once the PR is approved and merged into the main branch.

Another ADF Instance 002

We recommend adhering to the following naming conventions for branching:

- **Release branch:** ADF_Framework_<<month>>_Release
- **Development branch:** ADF_Sprint<<number>>_<<developer_username>>
- **Testing branch:** ADF_Framework_<<feature_title>>_testing

This Instance was created to separate out some independently running utility pipelines and triggers, which are not there in Instance 001 where main KDP ADF framework is deployed and running. It has the following pipelines:

Historical Data Migration Pipelines

It has 2 pipelines- 'PIPE_hist_data_migration_redshift_snowflake_child' and 'PIPE_hist_data_migration_redshift_snowflake_master'.

These pipelines are used to do historical load of data from redshift to snowflake. Based on the metadata tables for historical load, tables are loaded with historical data.

S3 to ADLS Pipelines

These utility pipelines are being used to bring the files from S3 to ADLS till the time Kenvue MBox is not enabled in production.

It has 3 pipelines- 's3_to_adls', 's3_adls_child', 's3_to_adls_date_folder_utility'.

's3_to_adls' and 's3_adls_child' are being used overall for this functionality. But 's3_to_adls_date_folder_utility' is only being used for Indonesia, where additional requirement of current date folders was needed.

's3_to_adls_date_folder_utility' has been decommissioned in KDP now.

Github Deployment Pipelines

These pipeline are being used to execute any snowflake script for deployment purpose in production.

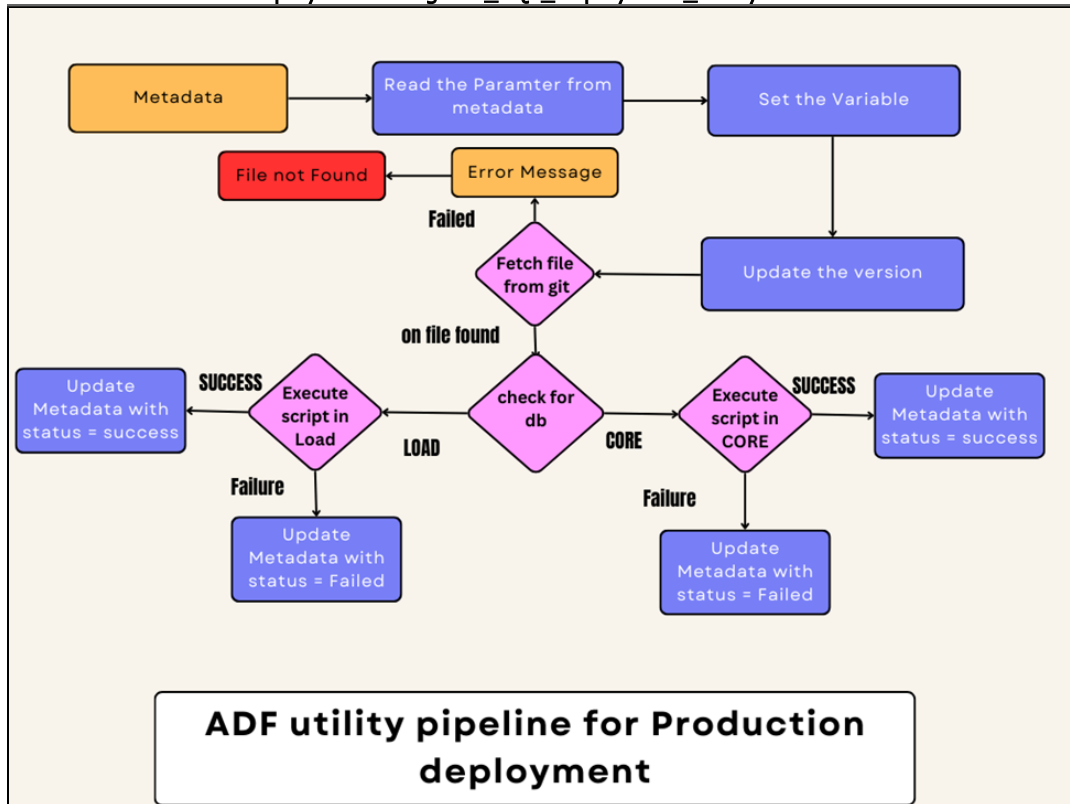
There are 2 pipelines ; 'runSQL' and 'GIT_SQL_Deployment_Utility'.

'runSQL' pipeline is casually being used by developers to run any quick script.

'GIT_SQL_Deployment_Utility' pipeline is majorly being used for any deployment. This pipeline picks up the latest script from github repo and execute it against snowflake prod.

(More information in next section)

The purpose of this pipeline is to deploy code changes (e.g., stored procedures, DDL, update statements, metadata) to the production environment. The pipeline is driven by metadata and requires specific parameters to run.



Metadata Table: PROD_RUN_METADATA

Columns used:

PROCESS_ID: Tracks the count of records.

DB: Indicates whether the change is for the Load DB or Core DB.

VERSION: Specifies the current version of the file.

FILE_NAME: Name of the file (format: version+.sql).

STATUS: Indicates the status of the run (Success / Fail).

TIMESTAMP: Logs the pipeline run time.

Pipeline Workflow

1. The pipeline reads the metadata table and retrieves the version from the previous run.
2. It then checks the Git repository for the next version of the file.
3. After each run (success or failure), the version number is incremented (depending on whether it's for Core DB or Load DB).
4. If the new version of the file is not found, the pipeline will not increment the version and will fail with a message indicating that the file is not located.

Steps to Use the Pipeline

1. Check the PROD_RUN_METADATA table for the previous version (for Load or Core DB, depending on which DB you wish to update).
2. Navigate to the Git repository (gitadfadbpl001) under the branch: DDL_DML_Scripts/prod_release_scripts/(core or load).
3. Depending on the target (Load or Core DB), click Add File in the relevant folder, and add the content to be executed in production.
4. Name the new file as the next higher version of the last executed file (e.g., if the previous version was 235.sql, the next should be named 236.sql).
5. Once the file is named according to the metadata versioning, trigger the GIT_SQL_Deployment_Utility.
6. Monitor the pipeline in ADF, or check the metadata table for the current execution status.