

Key2 Consulting

Data Ingestion Framework: ADF Pipeline Overview

**September 21, 2023**

**Table of Contents**

[Document Background 3](#_Toc146281522)

[Lakehouse Structure 3](#_Toc146281523)

[ADF Workspace Structure 3](#_Toc146281524)

[ADF Pipeline Methodology 4](#_Toc146281525)

[DIF SQL Server Pipelines 5](#_Toc146281526)

[DIF Delimited File Pipelines 6](#_Toc146281527)

[DIF Lakehouse Pipelines 6](#_Toc146281528)

[EIA API Pipelines 7](#_Toc146281529)

[EIA API Config File Parameters 8](#_Toc146281530)

# Document Background

This document describes the Azure Data Factory (ADF) pipelines used to ingest and transform data from on-prem and other sources into a lakehouse that resides in Azure Data Lake Storage (ADLS). These pipelines rely on the metadata and configuration information that is entered into the Data Ingestion Framework (DIF) database.

For more information on the fundamentals of the DIF, please consult the following documents:

* Key2 Data Ingestion Framework Overview
* Key2 Data Ingestion Framework - Configuration Database Guide

# Lakehouse Structure

A lakehouse is a concept that combines the elements of data lakes and data warehouses. The lakehouse is composed of three primary zones that logically organize data using a medallion architecture. As data flows through each zone, the structure and quality of the data is incrementally improved.

* **Bronze Zone** - The data is first landed into the Bronze. The data structure in this layer corresponds to the source system data structure. Source database table data and delimited files are stored as parquet files in the bronze zone.
* **Silver Zone** - The data in the Bronze Zone is then merged into the Silver Zone, where the data exists as a Delta Table (with underlying parquet files). The merge (upsert) logic can be either incremental or full, depending on how the target is configured in the DIF. The data in this layer has minimal transformations and supports ad-hoc reporting.
* **Gold Zone** – Using the Delta Tables in the Silver Zone, the data is then transformed into a denormalized data warehouse structure with dimension and fact tables that also utilize the delta format. The data in this layer is project specific, highly transformed and generally used to support specific reporting needs.
* **Landing Zone** – this layer is not officially part of the medallion architecture but serves as a staging area for source files that may first be acquired via processes like FTP or API. The data in this zone is stored using the original source file format and does not need to be persisted.

# ADF Workspace Structure

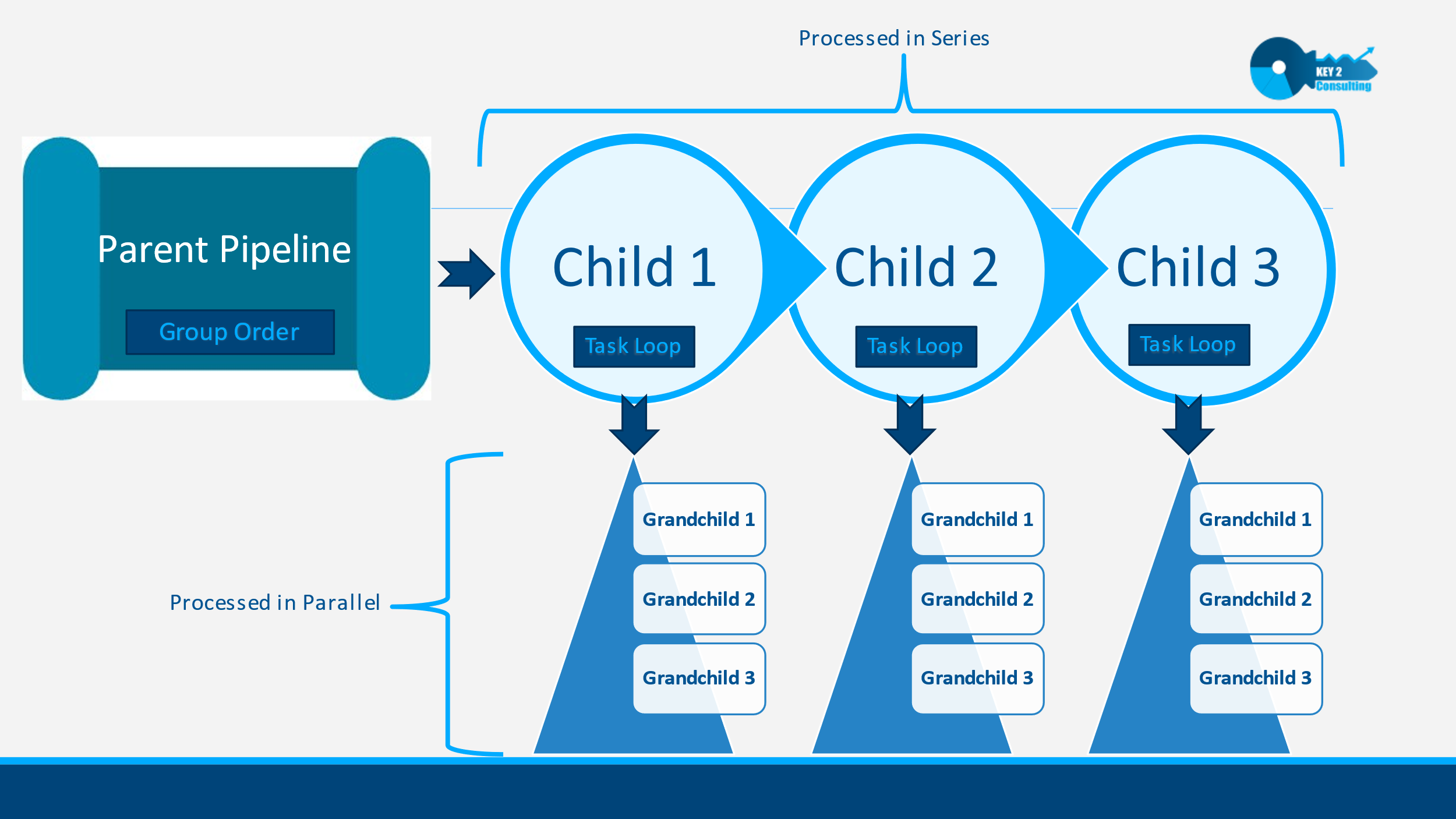
The ADF workspace is composed of folders with pipelines that are designed to process data from a specific type of data source. There is also a Common folder that contains functionality shared by the other three folders.

* **DIF SQLServer** – The pipelines in this folder pull data from an on-prem SQL Server database and move/transform the data into the Bronze and Silver Zones in the lakehouse.
* **DIF DelimitedFile** – The pipelines in this folder consume delimited text files and move/transform the data into the Bronze and Silver Zones in the lakehouse.
* **DIF Lakehouse** – The pipelines in this folder consume data in the Silver Zone and then perform the transformations needed to merge the data into the dimensional model in the Gold Zone.

# ADF Pipeline Methodology

The pipelines described above all follow a common parent-child-grandchild methodology. These pipelines also interrogate the DIF for the instructions needed to perform their specific tasks, which are summarized as follows:

* A **parent pipeline** (ending in “\_GroupOrder”) that manages the number and precedence of the data integration tasks configured in the DIF. For example, you may need to extract data from 100 SQL Server tables. Rather than having all 100 extracts run simultaneously, you can configure the DIF to divide the tasks into five execution groups containing 20 tables each. The execution groups are assigned a number between 1 and 5 and the parent pipeline will process each group in series by invoking 5 subsequent instances of the child pipeline.
* A **child pipeline** ( ending in “\_TaskLoop”) that determines which specific data integration tasks are in an execution group. The loop in this pipeline will spawn multiple threads that run in parallel. Continuing from the previous example, this pipeline would identify which 20 pipelines are in execution group X and then invoke 20 simultaneous instances of the grandchild pipeline.
* A **grandchild pipeline** (named either PL\_DIF\_SQLServer\_TableCopy, PL\_DIF\_DelimitedFile\_ToParquet or PL\_DIF\_Lakehouse\_Ingestion) that processes a single data integration task. This pipeline also queries the DIF to determine source/target configurations such as whether to perform a full or incremental load, watermarks, which columns to process and many more.

Assume the DIF is configured with three execution groups, each of which has three data integration tasks. The graphic below summarizes the ordering and precedence of these tasks.

# DIF SQL Server Pipelines

The pipelines in this folder are described as follows:

1. The **PL\_DIF\_SQLServer\_GroupOrder** pipeline is the parent pipeline that is executed first. It accepts two parameters:
   1. ProjectName
   2. DIGroupName
2. The **PL\_DIF\_SQLServer\_GroupOrder** pipeline queries the DIF using these two parameters to determine how many execution groups are configured for the given ProjectName/DIGroupName combination. If there are multiple execution groups configured, it will invoke one instance of the PL\_DIF\_SQLServer\_TaskLoop pipeline for each group. These instances will execute sequentially in order of the integer value assigned to the execution group.
3. The **PL\_DIF\_SQLServer\_TaskLoop** pipeline will then run for the first execution group. It queries the DIF to determine how many data integration tasks are in this execution group.
4. If there are multiple tasks configured in the group, the PL\_DIF\_SQLServer\_TaskLoop pipeline will invoke one instance of the **PL\_DIF\_SQLServer\_TableCopy** pipeline for each task. These instances will execute in parallel. The PL\_DIF\_SQLServer\_TableCopy pipeline copies the on-prem SQL Server table to a parquet file in the Bronze Zone of the data lake. This extract is performed by an ADF Copy Data activity and the DIF configurations determine if the extract is executed as either an incremental or full load.
5. The last pipeline to execute in this folder is the **PL\_DIF\_SQLServer\_RawToRefined** pipeline. This pipeline executes immediately after each instance of the PL\_DIF\_SQLServer\_TableCopy pipeline as a related data integration task. This task is performed via a generic, parameterized Databricks notebook that merges the data for the source table’s most recent parquet file in the Bronze Zone into a Delta Table in the Silver Zone.

# DIF Delimited File Pipelines

The pipelines in this folder are structured and function in the same manner as the SQL Server pipelines described above but consume delimited text files as their data source.

# DIF Lakehouse Pipelines

Like the DIF SQL Server pipelines, the **DIF Lakehouse** pipelines leverage the Group Order and Task Loop parent and child pipelines. However, the DIF Lakehouse grandchild pipeline utilizes customized logic to perform the transformation and loading of data into a dimensional data warehouse. This logic is stored in Databricks notebooks that are invoked at the grandchild level. Unlike the generic notebooks in the DIF SQL Server folder, there is one custom notebook for each data integration task that performs operations specific to that transformation step.

The execution groups must be configured in the DIF such that data integration tasks are performed in a sequence that supports dimensional modeling. For example, your exact implementation may vary slightly, but it should use a pattern like the following:

1. Transformations
2. Dimension Loads
3. Fact Loads

The pipelines in this folder are described as follows:

1. The **PL\_DIF\_Lakehouse\_GroupOrder** pipeline is the parent pipeline that is executed first. It accepts two parameters:
   1. ProjectName
   2. DIGroupName
2. The **PL\_DIF\_Lakehouse\_GroupOrder** pipeline queries the DIF using these two parameters to determine how many execution groups are configured for the given ProjectName/DIGroupName combination. If there are multiple execution groups configured, it will invoke one instance of the PL\_DIF\_Lakehouse\_TaskLoop pipeline for each execution group. These instances will execute sequentially in order of the integer value assigned to the execution group.
3. The **PL\_DIF\_Lakehouse\_TaskLoop** pipeline will then execute for the first execution group. It then queries the DIF to determine how many data integration tasks are in this group.
4. If there are multiple tasks configured in the execution group, the PL\_DIF\_Lakehouse\_TaskLoop pipeline will invoke one instance of the **PL\_DIF\_Lakehouse\_Ingestion** pipeline for each data integration task. These instances will execute in parallel. As described above, The PL\_DIF\_Lakehouse\_Ingestion pipeline invokes a custom Databricks notebook that performs operations specific to that target delta table and transformation step.
5. All tables created/refreshed in the DIF Lakehouse pipelines are stored as Databricks delta tables in the Gold Zone of the lakehouse.

# EIA API Pipelines

The EIA API pipelines contain the logic needed to extract data via API from the US Energy Information Administration (EIA). Key2 uses this API to acquire monthly electrical power generation data along with supporting data about utilities and power plants.

For more information on the API, please see the EIA’s official [documentation](https://www.eia.gov/opendata/documentation.php).

This folder is comprised of two pipelines:

1. **PL\_EIA\_Main** serves as the master package.
2. **PL\_EIA\_API\_Consume** serves as a generic pipeline that is used to make calls to any of the API’s endpoints. In other words, no endpoint specific configuration values are hard coded in this pipeline. Instead of using the DIF for configuration values, these pipelines leverage JSON config files, one for each API endpoint.

The EIA pipelines are summarized as follows:

1. The **PL\_EIA\_Main** pipeline is executed first and accepts two parameters:
   1. ParamFileSourceFolder – the full path to the storage account folder where the param file(s) are located.
   2. ParamFileCDS – a comma delimited string of param file names, one for each endpoint.
2. The **PL\_EIA\_Main** pipeline then splits the param file string and invokes one instance of the PL\_EIA\_API\_Consume pipeline for each param file.
3. The **PL\_EIA\_API\_Consume** pipeline performs the following tasks:
   1. Reads the param file to determine needed API endpoint, start month, end month, batch size, landing zone path and other settings pertinent for calling the API endpoint for a desired time frame.
   2. Calls the API end point for the desired timeframe. If multiple years are spanned, a separate parallel thread is invoked for each calendar year.
   3. If the config file specifies an incremental load, the logic will utilize a high watermark in the param file to dynamically determine the needed time frame. This watermark will also be updated in preparation for the next run after the API calls are completed.

# EIA API Config File Parameters

JSON configuration files are used to store API parameters for each API endpoint that needs to be called. A description of each parameter in the JSON file is provided in the table below.

|  |  |  |
| --- | --- | --- |
| Parameter Name | Description | Example |
| BaseURL | The URL path that is common to all API endpoints. This path will be appended with param values needed to call a specific endpoint, time frame, batch size, etc. | https://api.eia.gov/v2/ |
| MainAPIRoute | The parent dataset name. | Electricity |
| ChildAPIRoute | Parent datasets have child datasets, which may have children of their own. | facility-fuel |
| Frequency | The periodicity of measurements returned by the API. | monthly |
| DataPointList | A comma delimited string of the specific columns that should be returned in the dataset. | average-heat-content,consumption-for-eg,consumption-for-eg-btu |
| UpdateWatermark | When set to N, the pipeline is configured to run in **historical** mode and will strictly use the StartDate and EndDate values provided.  When set to Y, the pipeline is configured to run in **incremental** mode. It will first call the API endpoint to determine if data exists after the specified EndDate and update the EndDate param if so. After the API calls are completed successfully, the StartDate and EndDate values are updated accordingly to prepare the pipeline for the next incremental run. | Y |
| StartDate | The StartDate parameter instructs the API only to return data on or after a specific point in time. | 2023-01 |
| EndDate | The EndDate parameter instructs the API only to return data on or before a specific point in time. | 2023-05 |
| APIBatchSize | Used to instruct the API on how to paginate results and defines the number of records that should be returned in a single API call. The maximum value allowed is 5000. | 5000 |
| SortColumn | Used to instruct the API on how to paginate results. This parameter supports a comma delimited string of the columns that should be used to sort results in a single API call. | period,plantCode,primeMover |
| SortDirection | Used to instruct the API on how to paginate results. The sort direction is applied across of the columns specified in the SortColumn param. | desc |
| APIRetrySleepTimeSec | If the API encounters an error in the try block that contains the API request, the logic will wait the specified number of seconds before retrying the API call. | 60 |
| APIMaxRetryAttempts | If the API response is a value other than 200, the API will retry the API call for the number of time specified in this parameter. | 10 |
| SecretScope | The name of the secret scope configured in the Databricks workspace that points to the Azure Key Vault that contains the API key. | kv-key2demo-dev-eus-001 |
| APISecret | The secret in Azure Key Vault that is configured with the API key. | key2demo-eia-api-secret |
| LandingMountPoint | The Databricks mount point that specifies the container in which the landing zone folder exists. | /mnt/landingzone |
| LandingFolder | The root folder in the landing zone where API response files are posted (one file per response). The pipeline will append the main route and child route to this path. | api\_batching |
| CopyToMountPoint | The Databricks mount point that specifies the container in which the copy-to folder exists. | /mnt/landingzone |
| CopyToFolder | After the API pipelines run successfully, the JSON response files will be copied to a folder from which another process will be used to deserialize the JSON and load to the appropriate lakehouse zones. | generation |