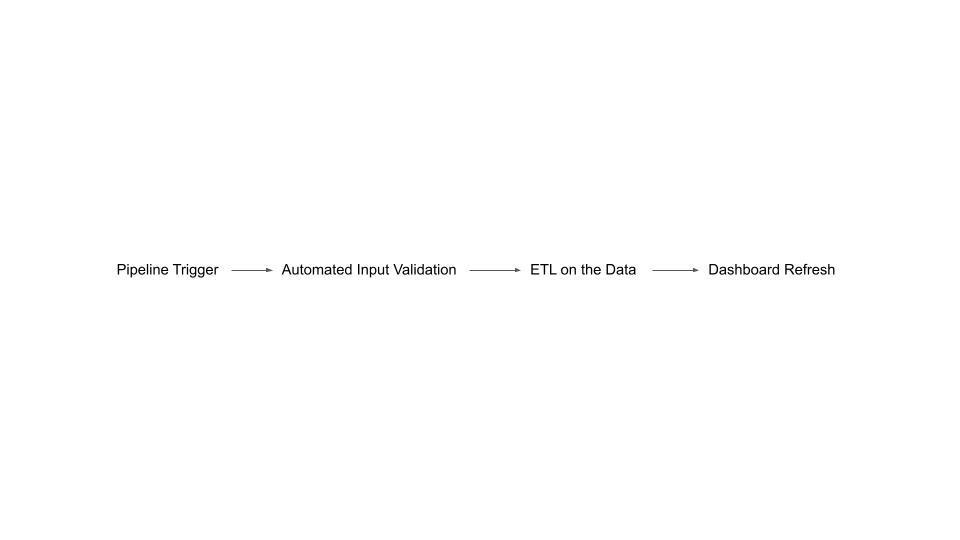
# End to End Pipeline Automation

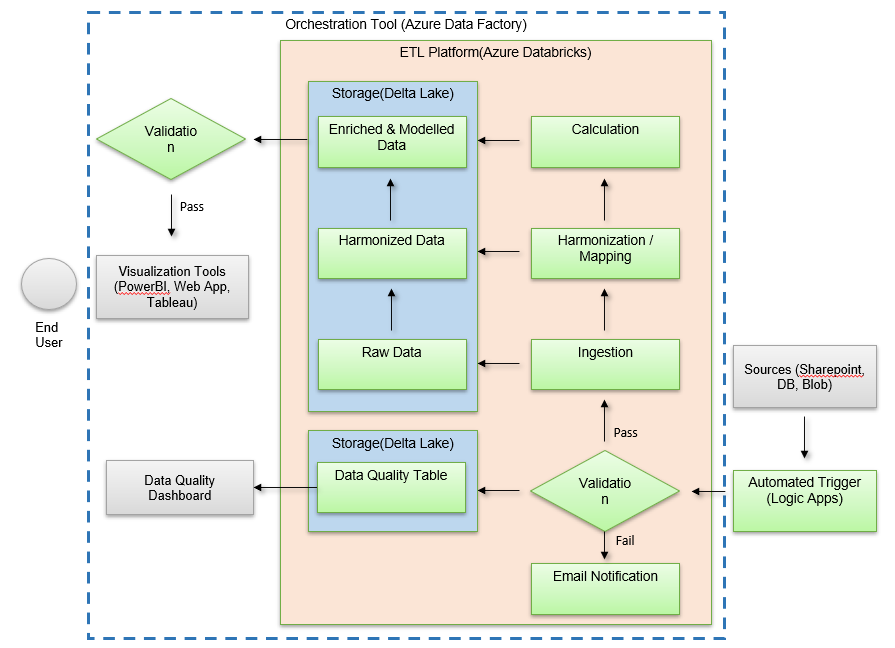
* Author: Azeem Azmat

End-to-end pipeline automation refers to the process of automating the entire data pipeline lifecycle, from data ingestion to visualization, to ensure seamless and efficient data processing and analysis. End-to-end pipeline automation streamlines the entire data processing lifecycle, from triggering the pipeline based on events or schedules, validating incoming data, ingesting it into the pipeline, harmonizing and processing it, to refreshing dashboards with updated insights. This automated approach improves efficiency, accuracy, and timeliness of data-driven decision-making processes, enabling organizations to derive maximum value from their data assets.

A comprehensive data integration and orchestration platform facilitates the automation and enhancement of data pipeline processes, leading to more effective data management, superior data quality maintenance, and accelerated insights generation. For instance, Azure Data Factory serves as a prime example, enabling users to automate and streamline the entire data pipeline lifecycle. Leveraging its capabilities, users can seamlessly process data, enhance data quality, and expedite insights delivery, thus empowering informed decision-making and optimizing the value extracted from data assets.

An automated end to end pipeline execution usually follows the below process flow.





## Step 1: Pipeline Trigger

Automatically triggering a pipeline depends on various factors such as your business requirements, data sources, processing needs, and system architecture. Here are some common scenarios when automatic pipeline triggering is beneficial:

* **Scheduled Processing:** If your data needs to be processed at regular intervals, such as daily, hourly, or weekly, automatic triggering based on a predefined schedule ensures timely and consistent execution of the pipeline.
* **Event-Driven Processing:** When data arrival or specific events trigger the need for processing, automatic triggering based on these events ensures that the pipeline reacts promptly to changes in data or system state. For example, processing new files uploaded to a storage system.

Automatic triggering facilitates integration with continuous integration and continuous deployment (CI/CD) pipelines. This allows for automated testing, deployment, and monitoring of data pipelines as part of the software development lifecycle.

### Tools:

Azure Logic apps can be used to trigger the pipelines when a file arrives in a specific folder of sharepoint.

#### What are logic apps?

Azure Logic Apps is a cloud-based service provided by Microsoft Azure that allows you to automate workflows and integrate various applications, data, services, and systems. It provides a visual designer to create workflows using a wide range of pre-built connectors and triggers without writing any code, although custom code can be integrated if needed. Logic Apps enable you to orchestrate and automate complex business processes across cloud and on-premises environments.

Key features and components of Azure Logic Apps include:

* **Triggers:** Triggers initiate the execution of a Logic App workflow. Triggers can be based on various events such as a new file being added to a storage account, a new email arriving in an inbox, an HTTP request, etc.
* **Connectors:** Logic Apps offer a wide range of connectors to interact with various services, including Azure services (like Blob Storage, SQL Database, Service Bus, etc.), third-party SaaS applications (like Salesforce, Office 365, Twitter, etc.), and on-premises systems (through Azure On-premises Data Gateway).
* **Actions:** Actions are the individual steps or tasks within a Logic App workflow. Actions perform specific operations such as sending an email, transforming data, calling an API, updating a database, etc.

Utilizing trigger activities in Azure Logic Apps, such as the "When a file is created or modified (properties only)" trigger, enables you to monitor a folder in a storage account for the arrival of new files or modifications to existing ones. Once a file is detected, you can initiate subsequent actions in your workflow, such as calling an HTTP endpoint or triggering an Azure Data Factory pipeline run using the appropriate connectors.

## Step 2: Automated input validations

Automated file validation is a crucial aspect of data quality assurance in any data pipeline. This manual will guide you through setting up an automated file validation process on Excel file against predefined expectations, checks column existence, data types, and specific value ranges, and sends email notifications in case of validation failures using Great Expectations and Databricks.

### 1. Prerequisites

Before proceeding, ensure you have the following:

* Access to a Databricks workspace
* Python environment with Great Expectations installed
* Basic knowledge of Databricks notebooks and Great Expectations concepts

Please refer to attached links for [great expectations documentation](https://docs.greatexpectations.io/docs/home/).

### 2. Setting Up Great Expectations on Databricks:

#### Install Great Expectations:

- Open a Databricks notebook.

- Install Great Expectations using the following command:

pip install great\_expectations

#### Initialize Great Expectations:

- Run the following commands to initialize Great Expectations:

import great\_expectations as ge

#### Connect to Data Sources:

- Configure Great Expectations to connect to your data sources such as databases, data lakes, or files.

- Great expectation offers configuration with a wide range of data sources. You can be working with pandas or spark dataframes and use below code for connecting with Great Expectations.

df\_ge = ge.dataset.PandasDataset(df)

For further detail on set up and working of great expectations on databricks please refer to the [official documentation](https://greatexpectations.io/blog/gx-databricks-notebooks-a-powerful-alliance-for-validated-data-workflows) and the [youtube tutorial](https://youtu.be/Cr3_yoCdBjg?si=n6iW0FXDk_kkXbAl).

### 3. Input File Name and Sheet Validation:

It's important to check whether the input file you received is in the aligned format. We shall create two checks for it when working on .xlsx files.

1. Check if the file name is correct. For example, sometimes we are getting month and year information from the file name. In such cases it's essential to detect such issues early.
2. Check if the sheet names in the excel are correct. Below is attached a sample case for checking such information.

if "ABC" in file\_name:

If "xyz" in sheet\_names:

df = xls\_book.parse(sheet\_name)

else:

dbutils.notebook.exit("Sheet Name not found")

else:

dbutils.notebook.exit("Not Agency Source File")

### 4. Great Expectations Validation:

Automated validations are usually dependent on use cases. While working on a wide range of data we have diverse expectations from our data which can be checked using the broad gallery of [Great Expectations](https://greatexpectations.io/expectations). Please keep in mind that automated validations should result in 3 possible scenarios:

1. Validation was a success
2. Validation failed but it should not stop the pipeline
3. Validation failed and it should stop further processing of the pipeline

Some of the important expectations which are applicable to almost all the use cases are listed below.

- Check the existence of the essential columns in your data set.

df\_ge.expect\_column\_to\_exist(

column,

result\_format={

"result\_format": "COMPLETE",

"return\_unexpected\_index\_query": True,

},)

- Check the data type of columns in your data set.

df\_ge.expect\_column\_values\_to\_be\_of\_type(

column=column,

type\_=type\_,

result\_format={

"result\_format": "COMPLETE",

"return\_unexpected\_index\_query": True,

},)

- Check the range of numeric columns in your data set to highlight outliers.

df\_ge.expect\_column\_values\_to\_be\_between(

"Share %",

min\_value=0,

max\_value=1,

result\_format={

"result\_format": "COMPLETE",

"return\_unexpected\_index\_query": True,

},)

- Check the columns whose values are expected not to be outside a certain set.

df\_ge.expect\_column\_values\_to\_be\_in\_set(

"Country",

["India", "USA", "Canada", "Mexico", "Brazil"],

result\_format={

"result\_format": "COMPLETE",

"return\_unexpected\_index\_query": True,

},)

- Check the columns whose values are expected not to be null.

df\_ge.expect\_column\_values\_to\_not\_be\_null(

column=column,

result\_format={

"result\_format": "COMPLETE",

"return\_unexpected\_index\_query": True,

},)

### 5. Handling Validation Results:

Great expectations provides cumulative results of all the expectations in a JSON format.

result = df\_ge.validate()

These validation results can be converted to a dataframe and logged for future reference.

#### Table Schema

To optimize data quality assessment and decision-making, we recommend incorporating specific columns in the validation results table. These columns offer distinct advantages:

| **Columns** | **Description** |
| --- | --- |
| Expectation\_Type | Clearly defines the nature of validation checks, aiding in the identification of data quality issues such as column existence, datatype conformity, or specific value presence. |
| Column | Provides contextual information on validated data attributes, enabling targeted analysis and remediation efforts by pinpointing areas of concern within the dataset. |
| Success | Swiftly communicates validation outcomes, facilitating efficient assessment of overall data quality status and differentiation between successful validations and areas requiring attention. |
| Unexpected\_Count | Quantifies the extent of unexpected occurrences during validation, offering quantitative insights into data anomalies and aiding in prioritizing remediation efforts based on severity and frequency. |
| Partial\_Unexpected\_List | Furnishes a detailed inventory of unexpected values encountered, empowering stakeholders to delve deeper into specific anomalies and facilitate targeted corrective actions. |
| Unexpected\_values\_weightage | Assigns weighted measures to unexpected values, enabling prioritization of resolution efforts based on the significance of deviations, thus enhancing decision-making by highlighting critical data integrity concerns. |
| Sheet\_Name | Specifies the name of the sheet within the Excel file that was checked. |
| File\_Name | Indicates the name of the Excel file that was checked. |
| Run\_Time | Shows the timestamp when the check was executed. |
| Spoc\_Name | Specifies the name of the person responsible for overseeing the check. |
| Email | Provides the email address of the person responsible for overseeing the check. |

By including these columns, stakeholders can efficiently assess data integrity, identify improvement areas, and enhance overall data quality and reliability.

#### Creating Dataframe from great expectations result:

Following code can be used to create the dataframe with above mentioned schema

result\_dict = result.to\_json\_dict()

Final\_df = gx\_func.dic\_to\_df(result\_dict,Sheet\_Name,File\_Name,Spoc\_Name,Email)

def dic\_to\_df(validation\_results,Sheet\_Name,File\_Name,Spoc\_Name,Email):

expectation\_data = []

for result in validation\_results['results']:

expectation\_config = result['expectation\_config']

expectation\_type = expectation\_config['expectation\_type']

column = expectation\_config['kwargs']['column']

unexpected\_count = result.get('result', {}).get('unexpected\_count', 0)

partial\_unexpected\_list = result.get('result', {}).get('partial\_unexpected\_list', [])

success = result.get("success", False)

unexpected\_percent\_total = result.get('result', {}).get('unexpected\_percent\_total', 0)

expectation\_data.append({

'Expectation\_Type': expectation\_type,

'Column': column,

'Success':success,

'Unexpected\_Count': unexpected\_count,

'Partial\_Unexpected\_List': partial\_unexpected\_list,

'Unexpected\_values\_weightage': unexpected\_percent\_total

})

df = pd.DataFrame(expectation\_data)

df['Sheet\_Name'] = Sheet\_Name

df['File\_Name'] = File\_Name

df['Run\_Time'] = datetime.now()

df['Spoc\_Name'] = Spoc\_Name

df['Email'] = Email

return df

#### Categorizing Failed Expectations

A reasonable approach to categorize errors and warnings is based on the nature and severity of the data quality expectations.

- Errors are critical issues that break the pipeline run.

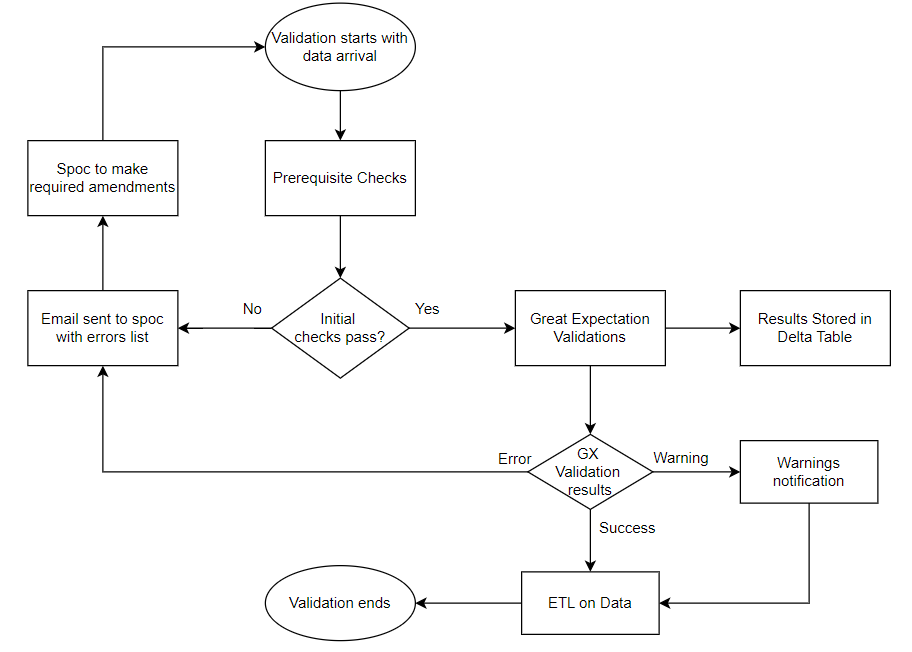
- Warnings are non-critical issues that do not break the pipeline run but should be logged for review.

| **Errors** | **Warnings** |
| --- | --- |
| Missing mandatory columns | Value Range Expectations |
| Mandatory data missing | Marginally deviating data |
| Invalid column formats | Marginally out-of-range data |

### 6. Automate Validation Process:

Schedule the validation process to run periodically based on the application requirement using orchestration tools like Azure Data Factory.

Handle validation failures by sending alerts or notifications. Monitor validation results and make adjustments as needed based on evolving data requirements.



Automating file validation using Great Expectations and Databricks streamlines the data quality assurance process, ensuring that incoming data meets predefined expectations. By following this manual and customizing the provided code, you can establish a robust and efficient validation framework for your data pipelines.

### 7. Validation Dashboard

For a validation dashboard aimed at monitoring data quality, particularly in the context of automated file validations, the following KPIs are crucial:

#### Total Files Processed:

Description: The total number of files that have been validated over a specific period.

Purpose: To provide an overview of the volume of data being processed.

#### Validation Success Rate:

Formula: (Number of Successful Validations / Total Files Processed) \* 100

Description: The percentage of files that passed all validation checks.

Purpose: To measure the overall success rate of data validations.

#### Error Rate:

Formula: (Number of Files with Errors / Total Files Processed) \* 100

Description: The percentage of files that contain critical errors.

Purpose: To identify the prevalence of critical issues that break the data pipeline.

#### Top Validation Errors:

Description: List of the most common validation errors encountered.

Purpose: Helps prioritize fixes for the most frequent issues.

#### Data Dirty Score:

##### Missing Values Score:

The proportion of mandatory data missing in the dataset.

##### Invalid Format Score:

The percentage of data entries that do not conform to the expected format or type.

##### Outliers Score:

The number of data points that fall outside the acceptable range.

By tracking these KPIs, you can ensure comprehensive monitoring of data quality and quickly identify areas that need improvement, thus maintaining the integrity and reliability of your data pipelines.

## Step 3 : ETL on the Data

- - - - - To be filled - - - - -

## Step 4: Dashboard Refresh

Dashboard refresh is a crucial step in ensuring that stakeholders have access to up-to-date insights and analytics derived from processed data.

### Scheduled Refresh:

Dashboards can be scheduled to refresh at specific intervals, such as daily, hourly, or on-demand, depending on the frequency of data updates and business requirements.

### Auto Refresh:

Dashboard refresh via API allows for programmatic triggering of dashboard updates, providing flexibility and automation in the dashboard refresh process. Here's how it works:

#### API Endpoint:

Dashboard refresh functionality is exposed through an API endpoint provided by the dashboarding platform. This endpoint typically accepts HTTP requests with parameters specifying the dashboard to be refreshed and any additional options.

#### Authentication:

Authentication mechanisms, such as API keys or OAuth tokens, may be required to access the dashboard refresh API. This ensures that only authorized users or applications can trigger dashboard updates.

For Example: A POST request can be made to [URL](https://login.microsoftonline.com.rproxy.goskope.com/common/oauth2/token) with content body:

grant\_type=password&scope=openid&resource=https://analysis.windows.net/powerbi/api& client\_id= 65c4ea7e-cf39-475b-ac38-330f590da19e&username=[abc@xyz.com](mailto:abc@xyz.com)&password=xyz

To authenticate and access refresh api.

#### Request Parameters:

The API request may include parameters specifying the dashboard to be refreshed, such as the dashboard ID or name, as well as any additional options or filters.

#### Triggering Refresh:

Sending a POST request to the dashboard refresh API triggers the refresh process. The API endpoint processes the request, retrieves the latest data from the underlying data sources, and updates the dashboard accordingly.

For Example: A POST request to [URL](https://api.powerbi.com/v1.0/myorg/groups/%7Bgroup_id%7D/datasets/%7Bdataset_id%7D/refreshes) will refresh the dashboard against the group id and dataset id provided in the URL.