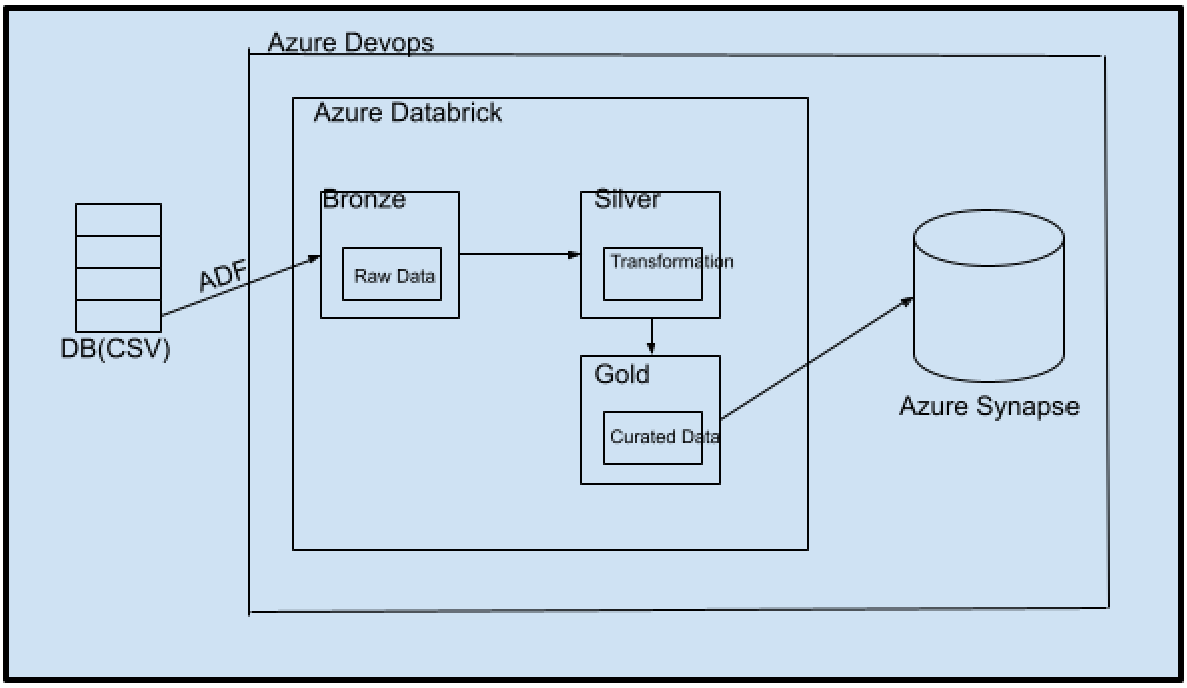
Data Pipeline for Colibra

Introduction

This project focuses on building a scalable and testable data pipeline to process wind turbine data for a renewable energy company. The pipeline follows the medallion architecture with three layers: raw, curated, and gold, and incorporates various data engineering best practices such as CI/CD, version control, data governance, exception handling, and monitoring.

Pipeline Architecture



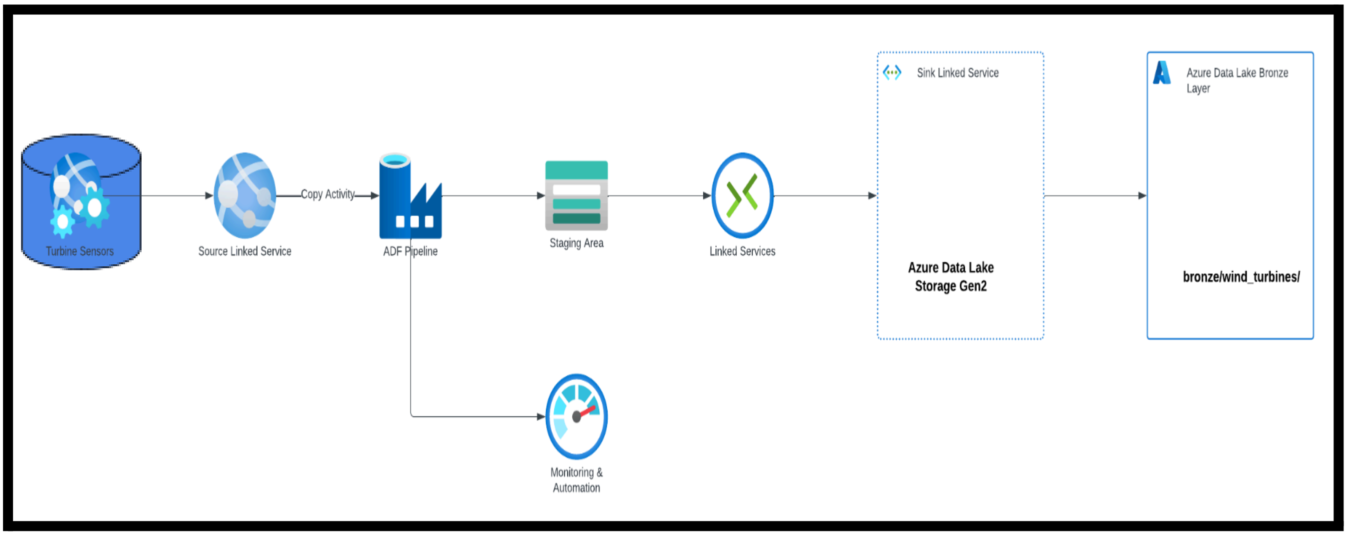
1. Raw Layer: Raw data is ingested using Azure Data Factory (ADF) and stored in an Azure Data Lake Storage (ADLS) container. This data is unprocessed and may contain missing values, outliers, and anomalies.

2. Curated Layer: Data is read from the raw layer and cleaned in Databricks using Spark. This involves handling missing values, outliers, and applying transformations. Summary statistics are calculated, and anomalies are identified and flagged.

3. Gold Layer: Cleaned and enriched data, along with summary statistics, is stored in Azure Synapse for reporting and further analysis. This layer supports dimension modelling and Slowly Changing Dimensions (SCD) for historical data tracking.

Data Ingestion (Raw Layer)

Data ingestion is handled by Azure Data Factory, which reads CSV files from the wind turbine sensors and stores them in the Raw Layer of ADLS. This happens daily as new data is appended to the CSV files.



Collect Data from DB:

- Data Generation: Each turbine group generates power output data, wind speed, and wind direction data in real-time. This data is batched and saved as CSV files (e.g., data_group_1.csv, data_group_2.csv, and data_group_3). Each file contains data from a specific set of turbines (e.g., turbines 1-5).
- Daily CSV Files: The turbine system produces these CSV files daily, each containing the last 24 hours of data for the corresponding turbine group.
- Sensor Connectivity: The sensors on the turbines are connected to a local on-premise server or an IoT hub, which transfers the CSV files to a staging area in the cloud or local storage.

Linked Services :

- Source Linked Service: Used in the Source Dataset of the Copy Activity to define where ADF should fetch the raw data from.
- Sink Linked Service: Used in the Sink Dataset of the Copy Activity to define where ADF should store the raw data in the Bronze Layer of Azure Data Lake.

Load Data to Azure Data Lake:

- ADF Pipeline : In Azure Data Factory (ADF), I create pipelines that orchestrate the movement of CSV files from the staging area (on-premise server, cloud folder, or blob storage) into Azure Data Lake.
- Pipeline Activities:
    - Copy Activity: The primary ADF activity is the Copy Activity, which moves CSV files to the Bronze Layer in Azure Data Lake. This layer is used for storing raw, untransformed data.

    - Source Dataset: In the Copy Activity, the Source Dataset points to the staging area where the CSV files are generated daily.
    - Sink Dataset: The Sink Dataset specifies the Azure Data Lake storage location in the Bronze Layer.

Monitoring Data Ingestion:

- ADF Monitoring Features: Azure Data Factory provides built-in monitoring capabilities, which allow it to track the status of each pipeline run. If any part of the ingestion process fails, alerts can be triggered for rapid intervention.

- Error Handling: The pipeline has error handling mechanisms to retry failed runs or alert the team in case of persistent failures (e.g., missing files due to sensor malfunction).

- Automation: Pipelines are scheduled to run at specific intervals (e.g., 24hrs) to ensure the latest turbine data is ingested and stored in the Bronze Layer.

Data Transformation (Curated Layer)

Once the raw data is ingested, the pipeline reads the data in Databricks, cleans it, calculates summary statistics, and detects anomalies.

- Cleaning: Handle missing values and outliers.
- Summary Statistics: Calculate min, max, mean for each turbine.
- Anomaly Detection: Flag turbines deviating by more than 2 standard deviations from the expected output.

Exception Handling

Exception handling ensures that any issues (missing files, incorrect formats) during ingestion or transformation are logged, and alerts are triggered.

Testing

Testing is integral to this pipeline. Unit tests are written to validate individual transformations, while regression tests ensure that new changes don’t break the existing logic.

Data Warehousing (Gold Layer)

The Gold Layer of the medallion architecture is where the cleaned and transformed data is stored for reporting and analysis. In this layer, the processed data is loaded into Azure Synapse Analytics, a powerful data warehousing solution that enables complex queries and analytics on large datasets.

Key Components of the Gold Layer

1. Data Storage:
   - The processed data, including the cleaned turbine data and summary statistics, is stored in structured tables within Azure Synapse.
   - This data is organised to facilitate efficient querying and reporting, often leveraging a star or snowflake schema for optimal performance.
2. Data Modeling:
   - Data in the Gold Layer is typically modelled to enhance performance for analytical workloads.
   - Dimension Tables: These tables store descriptive attributes related to the facts (e.g., turbine details, time dimensions).
   - Fact Tables: These tables contain quantitative data (e.g., power output, wind speed) and foreign keys to the dimension tables.
3. Slowly Changing Dimensions (SCD):
   - SCD is a method for managing and storing changes in dimension data over time. This is crucial for maintaining historical accuracy in reports and analytics.
   - There are several types of SCD, with Type 1 and Type 2 being the most common:
     - Type 1: Overwrites old data with new data. Historical information is not retained. This is suitable for non-critical information that does not require historical tracking.
     - Type 2: Creates a new record for each change, allowing for the preservation of historical data. This method adds a "version" or "effective date" field to track changes over time.
4. Implementation of SCD Type 2:
   - When loading data into the Gold Layer, if the turbine's attributes change (e.g., a new model or capacity), a new row is inserted in the dimension table with the updated information, and the previous record is marked as inactive (using an end_date or a current_flag column).

CI/CD Pipeline

Version Control

Codes, and ARM templates used to orchestrate the Azure resources, such as **Azure Data Factory**, **Azure Synapse** are stored in a Git repository (e.g., Azure Repos) for version control and collaboration.

Deployment

Azure DevOps pipelines is responsible for deploying the configurations that automate the movement of data from **on-premises SQL servers** to Databricks and Azure Synapse.

Security

Sensitive credentials such as **database connection strings** and authentication tokens for services like **Azure Synapse** are securely stored and accessed using **Azure Key Vault**

Orchestration

Orchestration involves coordinating multiple stages of deployment by defining jobs and tasks in YAML. The pipeline was configured to automatically trigger changes to the main branch, use self-hosted agents, and deploy Azure resources via ARM templates. This allowed seamless automation and control of workflows, optimising the deployment process and ensuring consistency across environments.

Monitoring & Alerting

Azure Monitor and **Log Analytics** are integrated to track the performance of data pipelines, including ingestion, transformation, and storage processes. These tools collect telemetry data to provide real-time insights into pipeline health. **Alerts** are configured to trigger notifications in case of failures, such as issues with data ingestion or transformations in Azure Data Factory, Azure Synapse, or other services. This ensures immediate attention is given to any disruptions, allowing for rapid troubleshooting and minimising downtime.

Data Governance

- Data Quality: Tools like Great Expectation, Deequ can be used for data validation and checks.
- Data Security: Encryption at rest and in transit using SSL, KMS, and key rotation.
- Audit: All actions are logged for auditability and traceability.

.

Logic

Exception handling

Testing

Overview

Testing in a PySpark-based data pipeline ensures the accuracy, consistency, and reliability of data as it moves through the pipeline. In Databricks, testing should be automated and integrated into your CI/CD pipeline to maintain code quality and detect issues early.

Unit Testing

Unit tests target individual transformations, functions, or logic within the pipeline. You test small pieces of code in isolation to verify they behave as expected.

Example:
Suppose you have a function that cleans data by removing null values. A unit test can mock a DataFrame, apply the transformation, and compare the result with the expected DataFrame.

CODE EXAMPLE:

```
import pytest

from pyspark.sql import SparkSession

from pyspark.sql import functions as F



def remove_nulls(df):

    return df.na.drop()



def test_remove_nulls():

    spark = SparkSession.builder.master("local").getOrCreate()

    input_data = [(1, "John"), (2, None), (3, "Doe")]

    input_df = spark.createDataFrame(input_data, ["id", "name"])



    result_df = remove_nulls(input_df)

    expected_data = [(1, "John"), (3, "Doe")]

            expected_df = spark.createDataFrame(expected_data, ["id", "name"])
    assert result_df.collect() == expected_df.collect()
```

In this case, you're ensuring that null values are correctly removed.

Regression Testing

Regression testing checks that updates or changes in the codebase do not break the existing functionality of the pipeline. This is crucial in large data pipelines where small changes in transformations or configurations can have unintended effects.

Example:
Suppose you’ve updated the logic for filtering out bad data. You can compare the output DataFrame from the new code version to the output from the previous version. Here, you may use snapshots of test datasets.

CODE EXAMPLE:

```
def test_filtering():

    old_version_output = load_snapshot("old_version.parquet")

    new_version_output = filter_bad_data(load_test_data())



    assert old_version_output.collect() == new_version_output.collect()
```

In this example, old_version_output is a snapshot of the output from the previous pipeline version, and you compare it with the output from the new version to ensure no unintended changes.

Tools for Testing in Databricks:

- Pytest: For organizing and running tests.
- Mocking Libraries: To simulate data and environment without actual dependencies.
- CI/CD Integration: Set up automated testing using tools like Jenkins, Azure DevOps, or GitHub Actions.

--CICD
Version Control
Deployment
Security (Azure Key Voult)
Orchistration
Monitoring
Alerting

Gold Layer:

Data warehousing

Tools: Azure synapse, Snowflake

Dimension Modelling
Scd /CDC Concept

Data Governance:
Data Quality Check(tools)
Data Security(ssl,kms, kts)
Authentication
Authorization
Encryption
Audit