NAME: - BALU LAKSHMI SUDHA

WEEK 5 CASE STUDY: - Smart Manufacturing Data Analytics Platform

Project Overview

The objective of this project is to ingest IoT sensor data, perform transformations, and validate the data using data quality checks with PyDeequ. The implementation has been done in two ways:

1. Databricks Jobs using DBFS storage.
2. ADF Pipeline using Azure Storage Account.

The project demonstrates orchestration, automation, and data quality validation for structured IoT data.

Data Description

The IoT dataset contains information about manufacturing plants and sensor readings. It includes a PlantID to identify the plant, a DeviceID to uniquely identify each IoT sensor, and a Timestamp representing the UTC time of the reading. Sensor measurements include Temperature in Celsius, Humidity in percentage, Vibration speed in mm/s, Pressure in bar, and Energy consumption in kWh. The Status column indicates operational status with values like OK, WARNING, or ALERT.

Method 1: Databricks Jobs

Step 1: Upload Data to DBFS

CSV files were uploaded to DBFS using the Databricks UI.

Example path:

dbfs:/FileStore/shared_uploads/traininguser8@sudosu.ai/IOT_sensor_data.csv

Data was read using Spark DataFrame

Step 2: Notebook Development

Created four notebooks to handle:

1. Data ingestion

2. Data transformation
3. Data quality validation using PyDeequ
4. Report generation

Widgets were used to pass parameters, allowing notebooks to process one file at a time.

Step 3: Data Quality Checks with PyDeequ

Installed Scala runtime and Deequ JAR libraries on the cluster.

Defined rules like:

- Completeness of columns (PlantID, DeviceID)
- Non-negative numeric values
- Allowed values for Status column (OK, WARNING, ALERT)

Results were written as JSON to DBFS.

Step 4: Job Orchestration

Created a Databricks job linking the four notebooks.

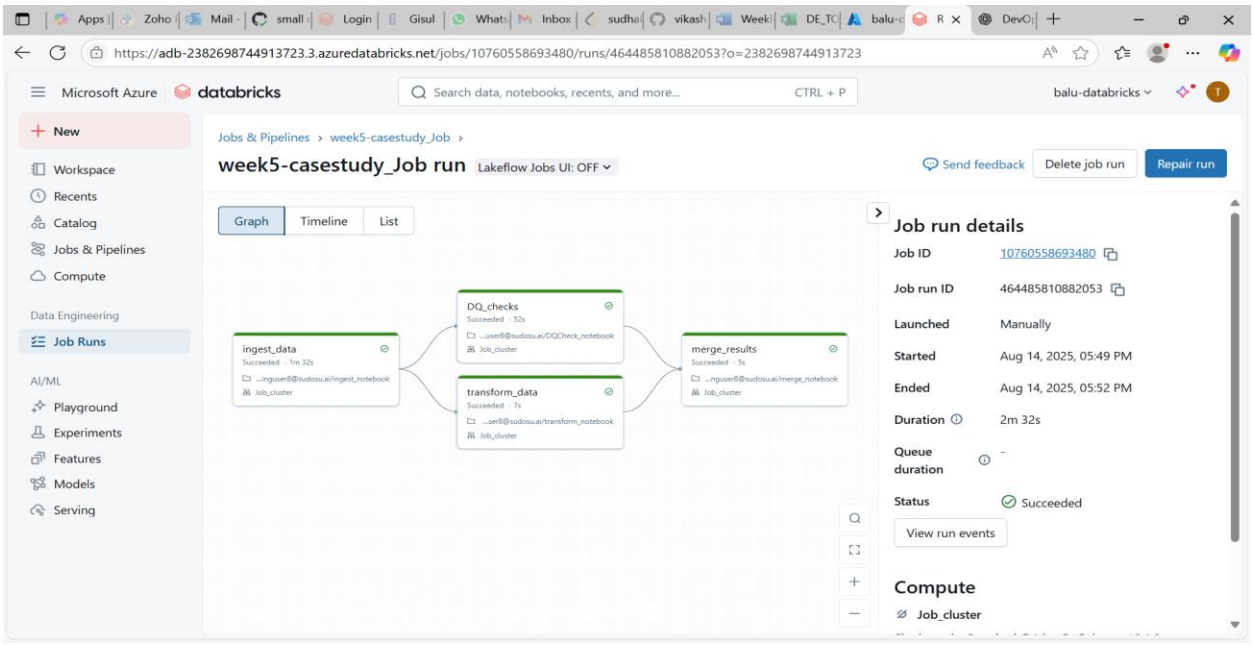
Implemented fan-out/fan-in workflow:

Fan-out: run multiple notebooks/tasks in parallel

Fan-in: combine results for final reporting

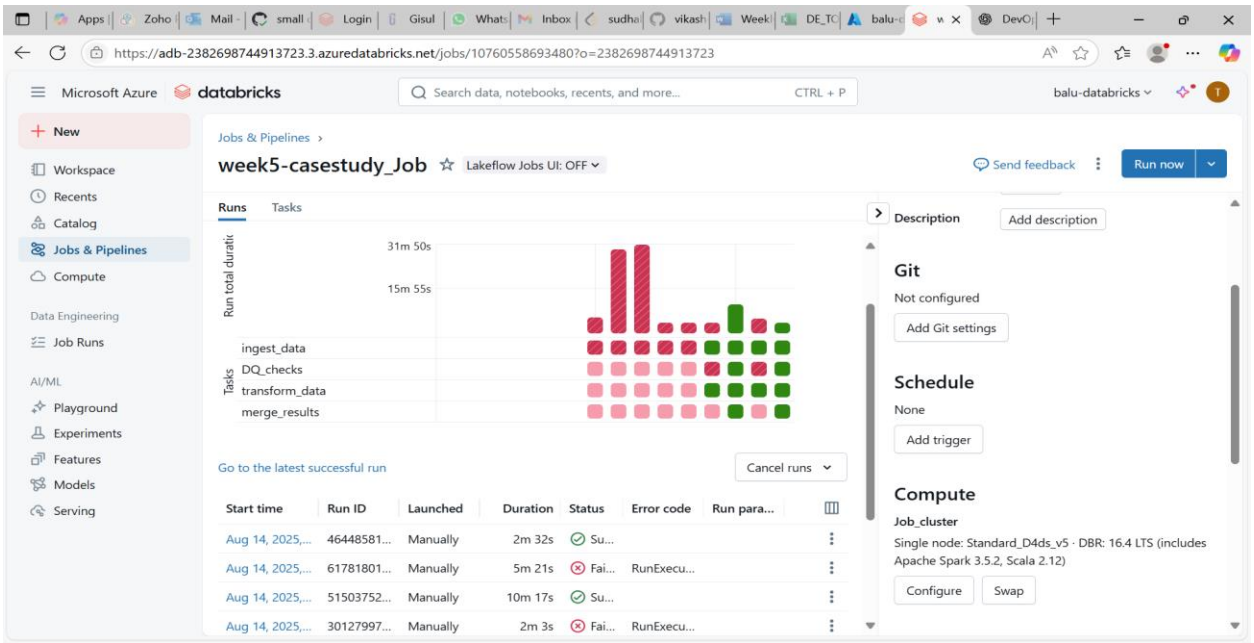
Jobs were executed manually and automated through GitHub Actions using a YAML workflow file.

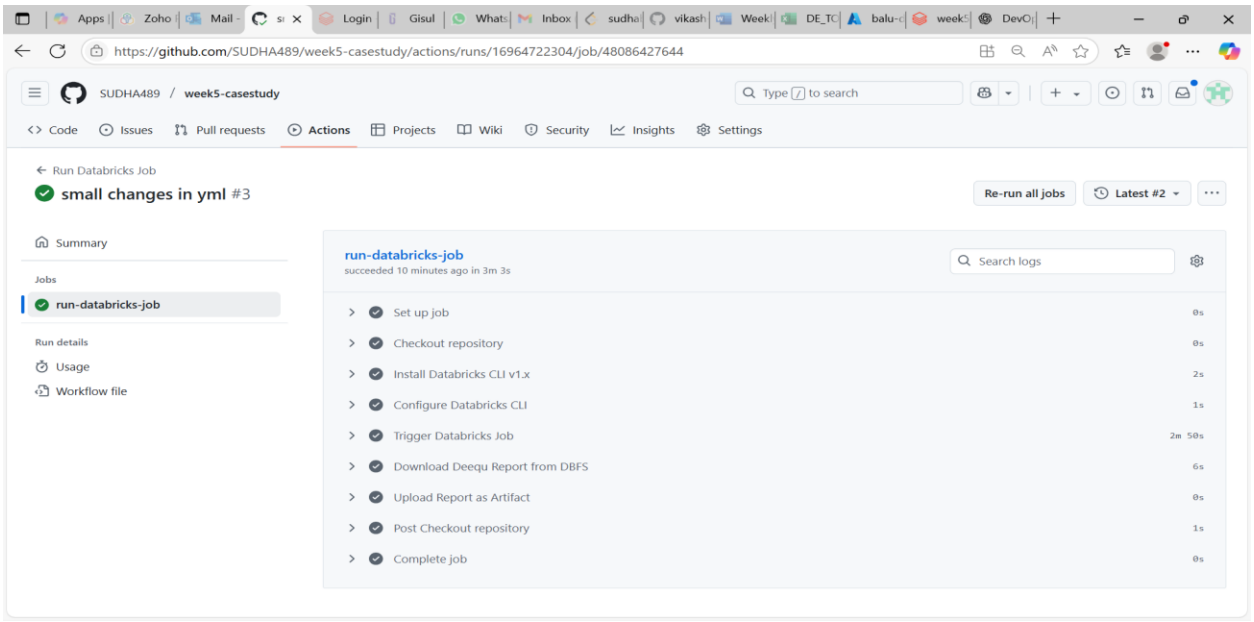
Output included Deequ JSON reports for validation.



Step 5: Observations

- Databricks jobs process one file at a time via widgets.
- DBFS storage is suitable for development and prototyping.
- GitHub Actions workflow successfully triggers the job but requires correct cluster setup and library installation.





Method 2: ADF Pipeline

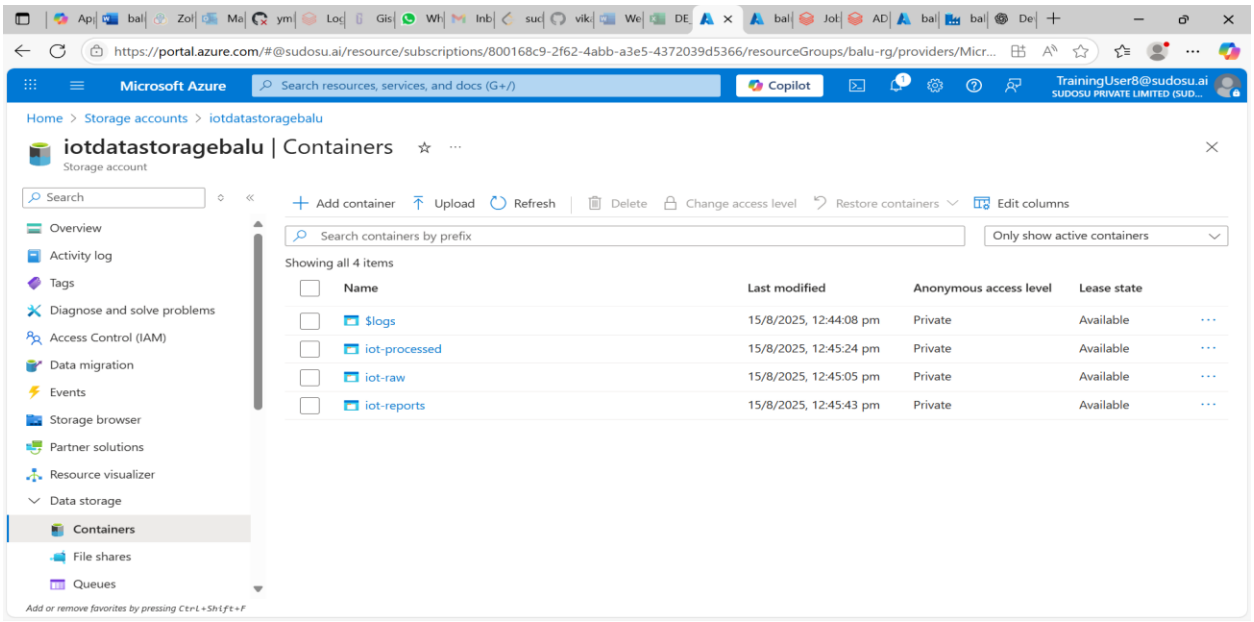
Step 1: Storage Setup

Created an Azure Storage Account with three containers:

iot_raw – store raw CSV files

iot_processed – store transformed data

iot_reports – store Deequ JSON reports



Step 2: Linked Services & Datasets

Created Linked Services for the storage account and Databricks workspace.

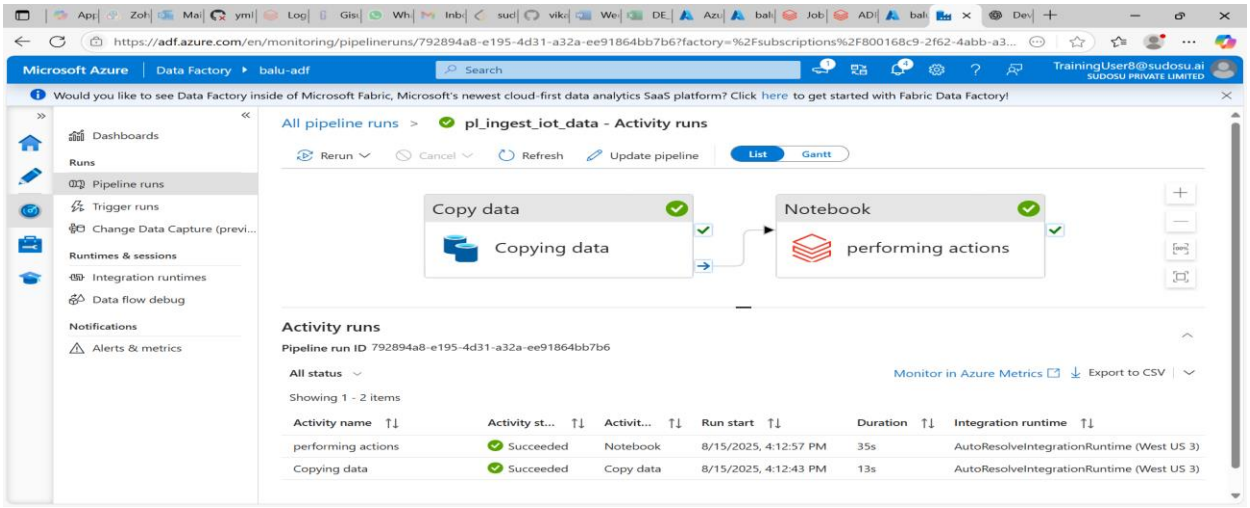
Created Datasets:

- Source: iot_raw
- Sink: iot_processed

Step 3: Pipeline Design

ADF pipeline consisted of:

- Copy Activity: Moves data from iot_raw → iot_processed
- Databricks Notebook Activity: Performs data quality checks on all files in iot_processed

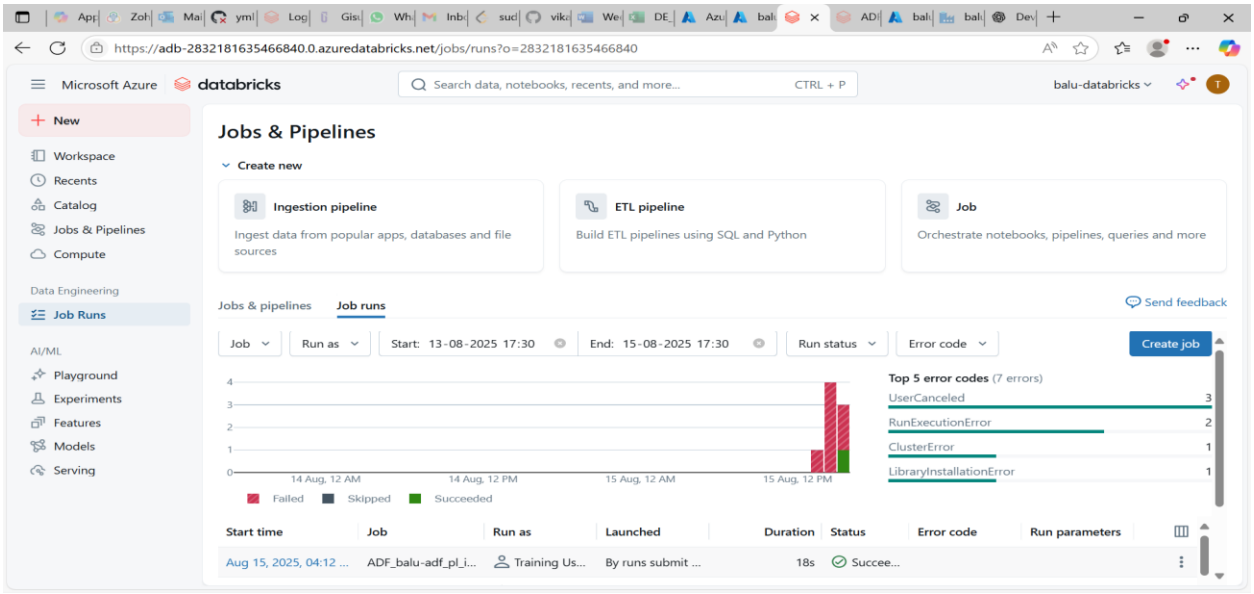


Step 4: Data Quality Execution

Notebook reads all files in the processed container using Spark.

Runs PyDeequ checks similar to Databricks jobs.

Writes JSON reports to iot_reports.



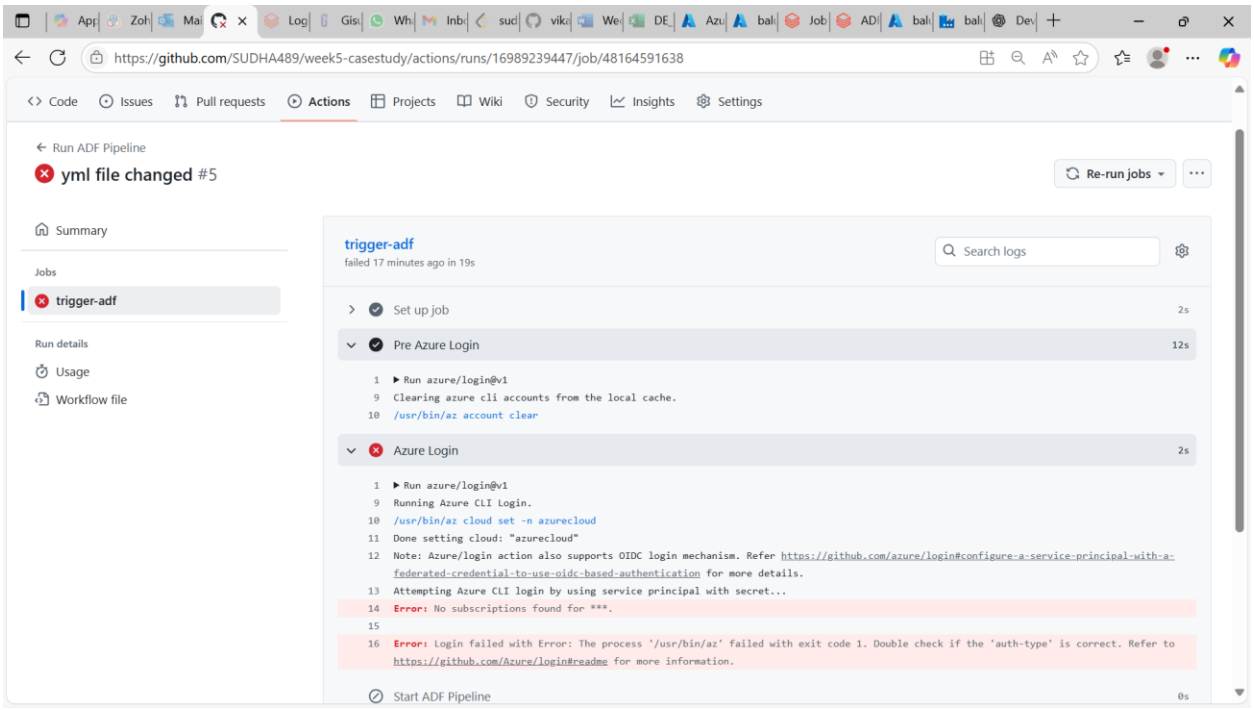
Step 5: Observations

Pipeline processes multiple files automatically, no need for widget parameters.

Requires Azure Storage credentials (SAS token or account key) for Spark to access data.

GitHub Actions can trigger the ADF pipeline using Azure CLI, but correct service principal permissions are required.

Pipeline is closer to a production-grade workflow.



Comparison Between Databricks Job & ADF Pipeline

In the Databricks job approach, DBFS storage is used, processing occurs one file at a time via widgets, and fan-out/fan-in orchestration is implemented within the job. Automation via GitHub Actions triggers jobs and outputs JSON reports to DBFS. In contrast, the ADF pipeline uses Azure Storage Account, automatically processes multiple files, and integrates a Databricks notebook for data quality checks. Automation via GitHub Actions is possible if the service principal has contributor permissions. Both approaches use PyDeequ for data quality checks, but Databricks jobs require Scala and JAR library installation, while the ADF notebook reads multiple files directly from storage.

Notes & Key Learnings

Databricks Jobs are ideal for development and testing, especially for single-file processing. ADF Pipelines provide automated, scalable, production-ready workflows for multiple files. PyDeequ requires proper library installation on the cluster. GitHub Actions can automate triggering, but service principal permissions are crucial. For ADF pipeline triggered via

GitHub Actions, permission errors may occur if the service principal does not have sufficient access.