ETL AND ML AUTOMATION

1. Introduction

In today's data-driven landscape, organizations are under increasing pressure to derive actionable insights from rapidly growing and diverse data sources. However, traditional approaches to data processing and machine learning development are often fragmented, manual, and difficult to scale. This not only introduces delays and operational inefficiencies but also creates challenges in maintaining data quality, governance, and reproducibility across environments.

To address these challenges, we have implemented a fully automated, cloud-native ETL and machine learning pipeline leveraging Microsoft Azure's unified data services. This project showcases an end-to-end solution that integrates data ingestion, multi-stage transformation, automated notebook orchestration, CI/CD for code deployment, and advanced analytics, all within a single cohesive ecosystem.

The pipeline is built around the Medallion Architecture (Bronze → Silver → Gold), ensuring clean separation of concerns, better data quality control, and support for analytics-ready datasets. It is also tightly integrated with Azure DevOps for version control and continuous delivery and is instrumented for observability using Azure Log Analytics to monitor health, failures, and performance bottlenecks.

Key Solution Highlights:

- **Data Ingestion from GitHub**: ADF pipelines are configured to automatically extract datasets from a public GitHub repository, enabling seamless integration with external data sources while ensuring flexibility and maintainability.

- **Centralized Storage in ADLS Gen2**: All ingested data is stored in Azure Data Lake Storage Gen2, organized across bronze, silver, and gold layers. This structured approach supports traceability, schema enforcement, and efficient downstream processing.

- **Data Processing via Azure Databricks**: Seven modular notebooks were developed to handle ETL and ML workflows. These notebooks are executed using scalable job clusters and leverage Apache Spark for parallelized transformation and analytics.
- **Workflow Orchestration using Azure Data Factory**: ADF acts as the central control plane, orchestrating the entire lifecycle—from data ingestion and storage to launching Databricks jobs for transformation and model execution.

- **CI/CD Automation through Azure DevOps**: All code and notebooks are version-controlled using Git and deployed via Azure DevOps YAML pipelines. This enables automated promotion of changes across development environments and ensures reproducibility of the pipeline.

- **Monitoring with Azure Log Analytics**: All Spark job executions, pipeline statuses, and cluster logs are collected in Azure Log Analytics, providing full transparency and enabling proactive issue detection and alerting.

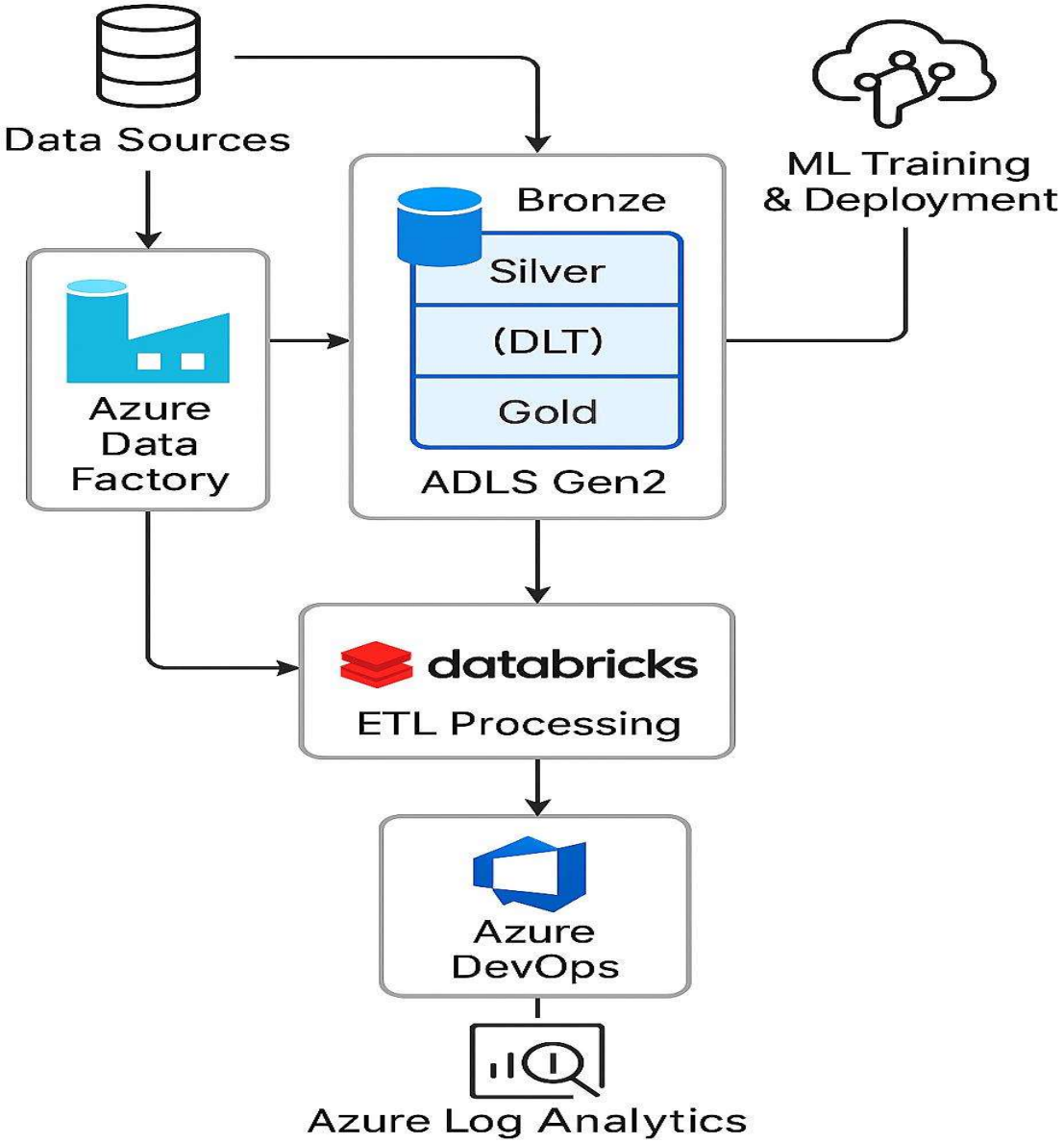
2.Problem Statement

The increasing volume and complexity of data require efficient, scalable, and automated workflows for data ingestion, transformation, and machine learning. Manual and fragmented data processes result in operational inefficiencies, data quality inconsistencies, delayed insights, and limited scalability. Furthermore, the absence of integrated automation and monitoring complicates management of data lineage, pipeline reliability, and reproducibility across environments.

This project addresses these challenges by implementing an end-to-end automated pipeline that leverages cloud-native services to enable reliable data ingestion, layered transformations, and machine learning, while ensuring automation, governance, observability, and scalability

3.Architecture Overview

This architecture implements a scalable, automated, and end-to-end data and machine learning pipeline leveraging Azure-native services and Databricks. It follows a medallion architecture pattern (Bronze, Silver, Gold) to ensure data quality, lineage, and progressive refinement, while supporting ML model development and deployment with automation and observability.



Key Components

1. **Data Sources:** Raw data originates from various external and internal systems, including transactional databases, IoT sensors, APIs, or third-party datasets, typically in formats such as CSV, JSON, or Parquet.

2. **Azure Data Factory (ADF):** Orchestrates and triggers ingestion pipelines on schedule or events, moving raw data securely into the cloud storage layer (ADLS Gen2). It supports parameterized pipelines for flexible ingestion and coordinates downstream processing workflows.

3. **Azure Data Lake Storage Gen2 (ADLS Gen2):** Serves as the centralized, scalable data repository, structured into:
    a. **Bronze Zone:** Raw, ingested data stored as Delta tables for reliability.
    b. **Silver Zone:** Cleaned, validated, and enriched data refined for business use.
    c. **Gold Zone:** Aggregated, highly curated data optimized for analytics and ML workloads.

4. **Azure Databricks:** Provides the compute platform and collaborative workspace to execute data transformations and machine learning workflows. Components include:
    a. **Bronze Processing:** Initial cleansing, schema enforcement, and storage of raw data as Delta tables.
    b. **Silver Processing:** Intermediate transformations and data quality checks using Delta Live Tables, with automatic lineage tracking.
    c. **Gold Processing:** Final aggregation and preparation of datasets for analytics and machine learning.

5. **Machine Learning Layer**
    a. ML models are developed, trained, and validated within Azure Databricks using the curated Gold datasets.
    b. Supports various ML frameworks and libraries (e.g., Spark MLlib, scikit-learn, TensorFlow).
    c. Model training pipelines leverage scalable compute resources and include hyperparameter tuning and cross-validation.
    d. Models can be registered, versioned, and deployed as REST APIs or batch inference pipelines.

6. **Model Management and Deployment**
    a. Model artifacts and metadata are managed within Databricks or integrated with Azure Machine Learning for advanced lifecycle management.
    b. Continuous Integration and Continuous Deployment (CI/CD) pipelines are automated via Azure DevOps, enabling seamless promotion of ML models and data pipelines from development to production environments.

7. **Delta Lake:** Ensures ACID compliance, schema enforcement, time travel, and efficient incremental data processing across all layers, improving data reliability and pipeline robustness.

8. **Azure Log Analytics:** Captures pipeline execution logs, data quality metrics, ML training statistics, and system health indicators. This observability layer supports alerting, troubleshooting, and governance.

9. **Azure DevOps:** Orchestrates automated CI/CD pipelines for code repositories containing Databricks notebooks, configuration files, and infrastructure as code. This facilitates version control, testing, and automated deployment across environments (Dev → QA → Prod).

4. Dataset Overview

The project utilizes the U.S. House Sales Dataset, which comprises comprehensive transactional and property-related data capturing residential real estate sales across the United States. This dataset forms the basis for data ingestion, transformation, and predictive modeling workflows implemented within the ETL and machine learning automation pipeline.

Dataset Description

The dataset includes detailed features such as sale price, property attributes (e.g., number of bedrooms, bathrooms, square footage), location identifiers (city, zip code), and temporal data (sale dates). These attributes enable effective exploration of market dynamics and support the development of regression models for accurate house price prediction.

Data Characteristics

- **Record Volume:** Approximately [specify number if known, e.g., 50,000+] individual house sales.
- **Data Format:** Tabular, structured as CSV/Parquet files ingested into the data lake.

Feature Category	Description	Data Type
Sale Price	Final transaction price of the house	Numeric (Target variable)
Sale Date	Date when the house was sold	Date (YYYY-MM-DD)
Location	Geographic information about the property	Categorical (City, State, Zip Code)
Property Size	Measurements related to house size	Numeric (Bedrooms, Bathrooms, Sq Ft)
Lot Size	Total area of the property lot	Numeric (Square feet or acres)
Year Built	Year the house was originally constructed	Numeric
Renovation Year	Year of last renovation or remodeling (if any)	Numeric or Null
Property Type	Classification of property type	Categorical (Single-family, Condo, etc.)
Condition & Grade	Quality and condition rating of the property	Categorical or Numeric scale
Garage/Parking	Garage size or number of parking spots	Numeric or Categorical

Basement	Presence and size of basement	Binary + Numeric
Heating/Cooling Type	Types of heating and cooling systems installed	Categorical

5. Creating Azure Data Lake Gen2 and Azure Data Factory Pipeline

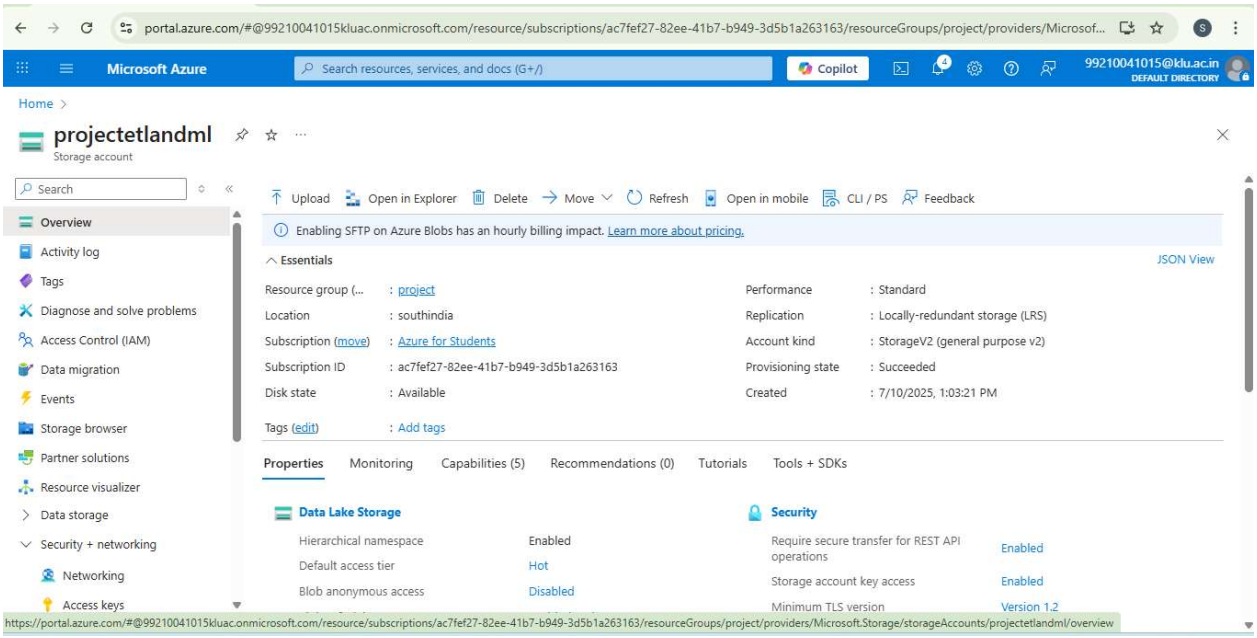
Overview

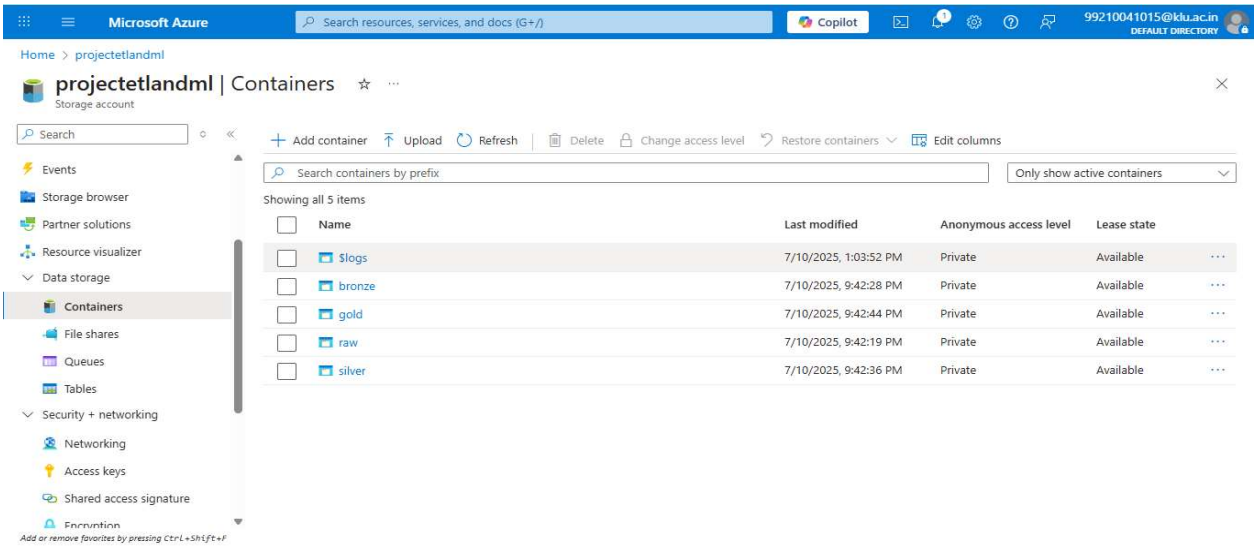
In this phase of the project, we set up the core infrastructure and automation workflows to enable seamless and automated data ingestion from GitHub into the Azure Data Lake Storage Gen2. This setup is essential for building a robust and scalable ETL pipeline that feeds raw data into the downstream analytics and machine learning processes.

Key Components and Workflow

- **Azure Data Lake Storage Gen2 (ADLS Gen2):**

ADLS Gen2 serves as the central repository (data lake) for storing raw and processed datasets in a scalable and secure manner. It supports hierarchical namespaces and is optimized for big data analytics.

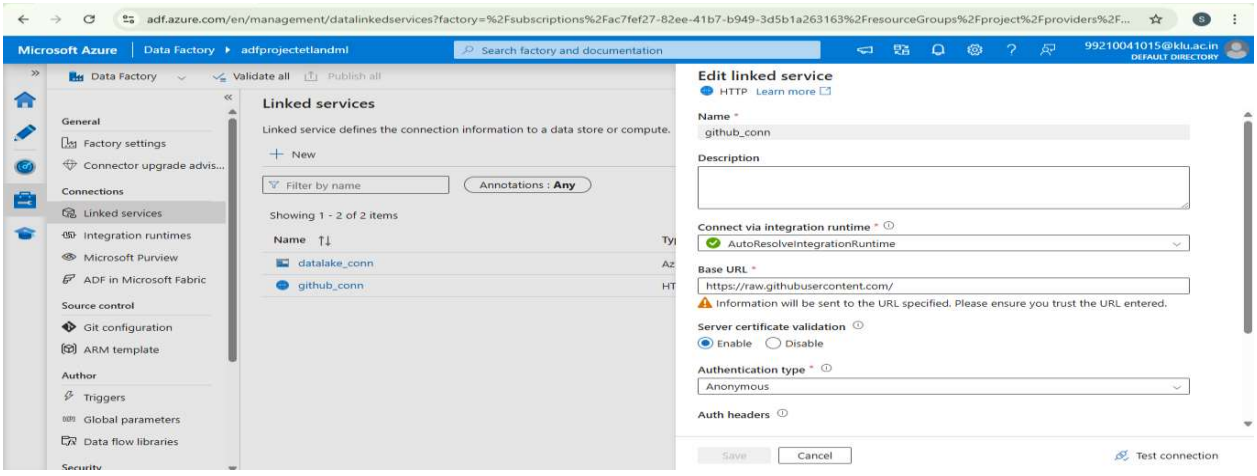




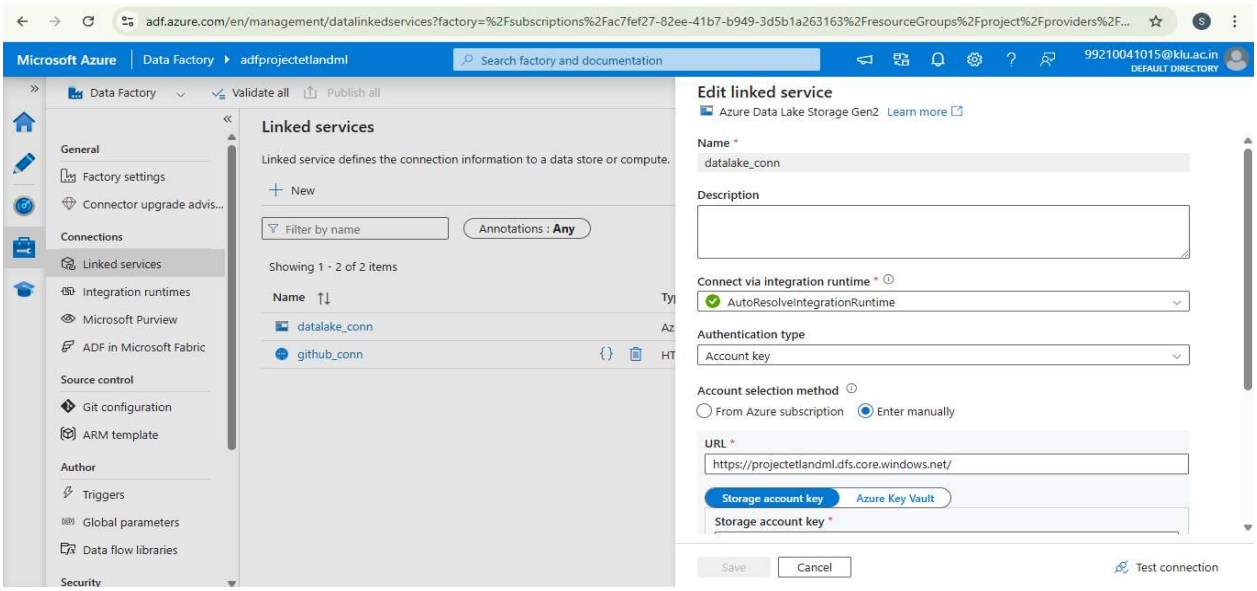
- **Azure Data Factory (ADF):** ADF is the orchestration service used to automate data movement and transformation workflows. It enables the creation of pipelines that connect multiple data sources and perform complex data operations.

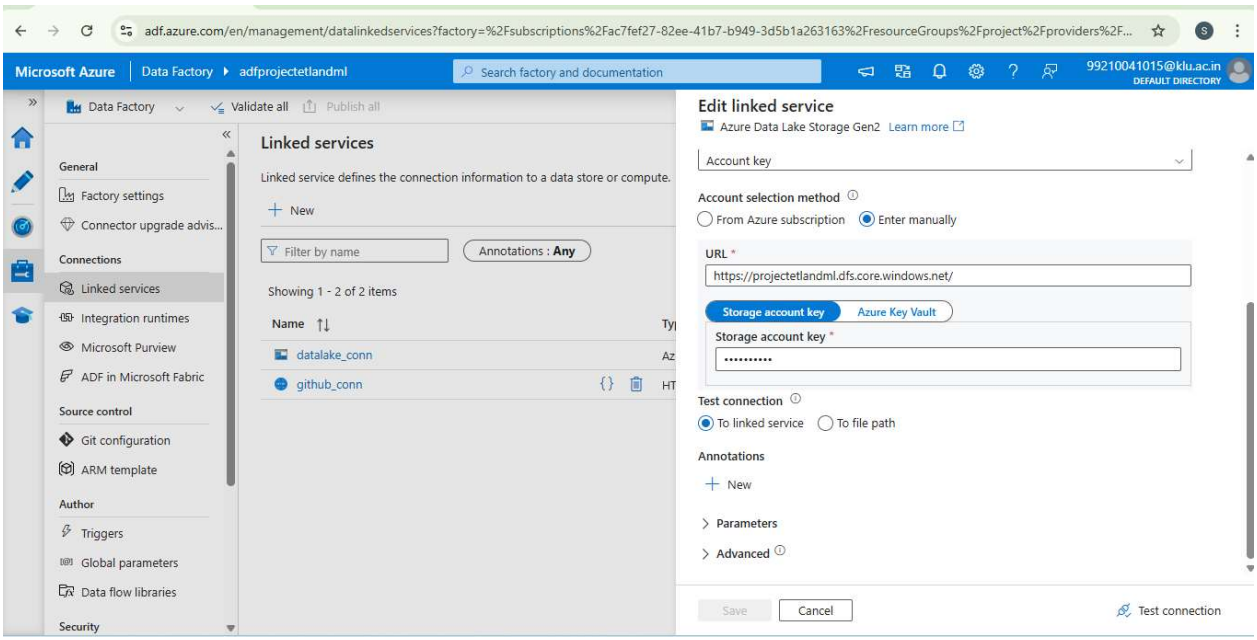
Pipeline Implementation Details

- **Linked Services Configuration:**
  - **GitHub Linked Service:** Configured to establish a secure connection between ADF and the GitHub repository hosting the source data files.



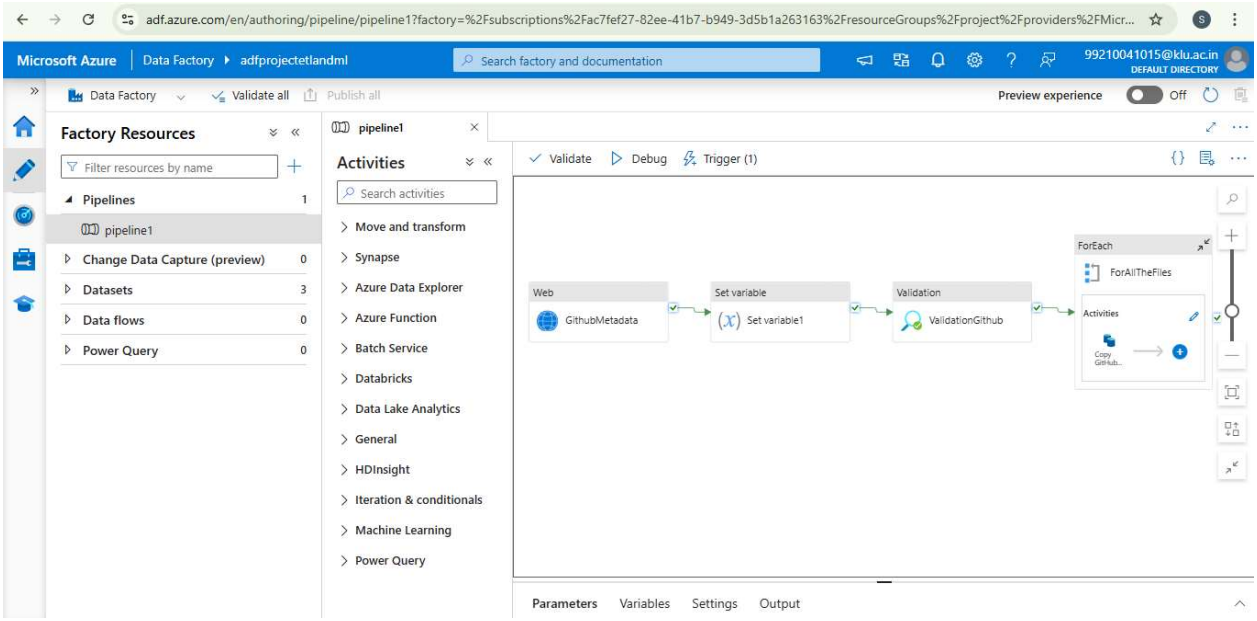
  - **Azure Data Lake Storage Linked Service:** Configured to connect ADF to the ADLS Gen2 account where data will be stored.

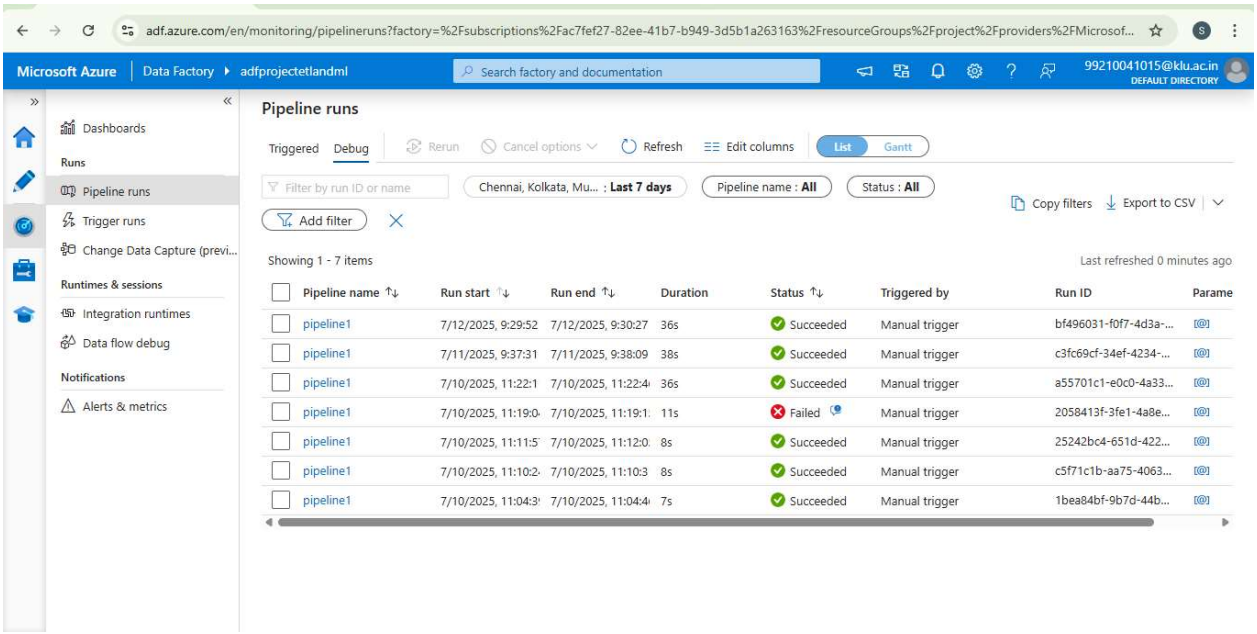




- **Pipeline Activities:**
  - **Copy Data Activity:** Copies raw data files from the GitHub repository to ADLS Gen2.
  - **ForEach Activity:** Iterates over multiple files or datasets from GitHub to ensure all relevant data is ingested.
  - **Validation Activity:** Checks the existence and validity of files before processing to avoid errors.
  - **Set Variable Activity:** Used to set and update pipeline variables dynamically for controlling flow and parameters.

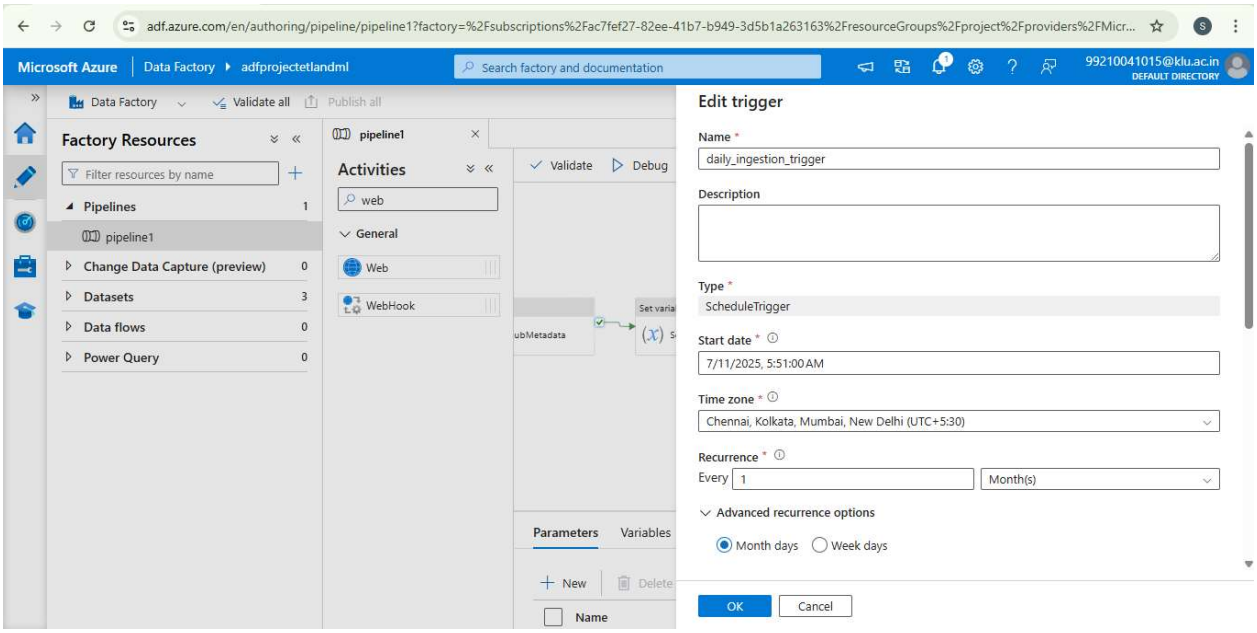
- **HTTP Connector:** Used for accessing GitHub API or fetching data via HTTP requests when required.

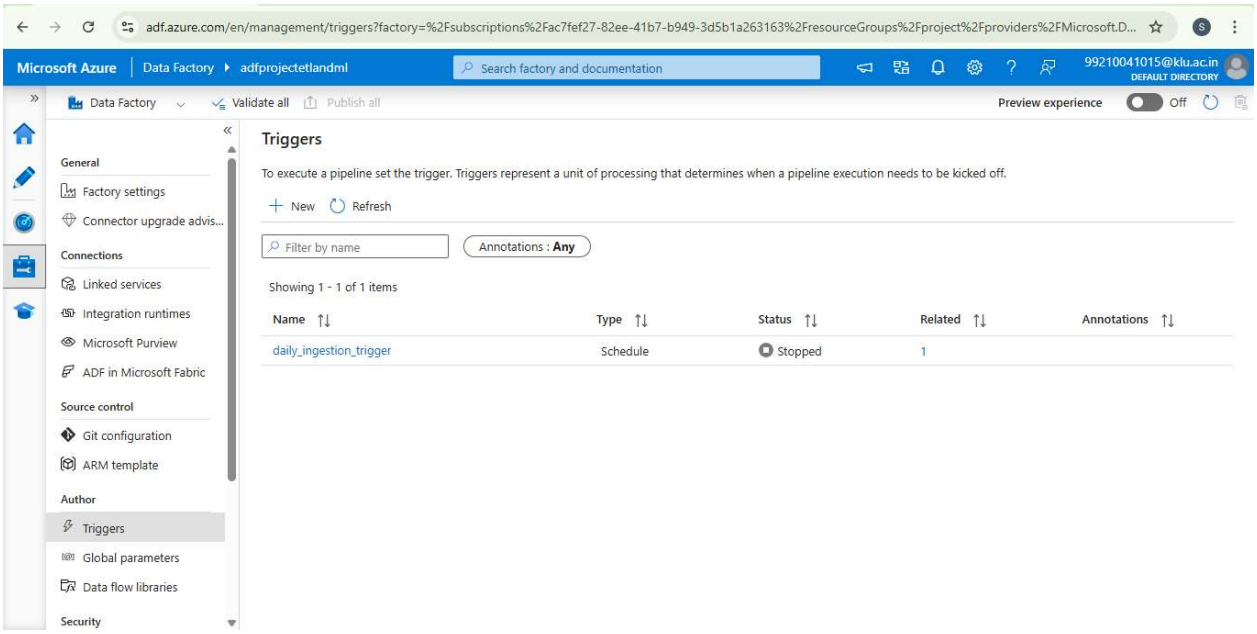




- **Trigger Configuration:**

An automated trigger is set up to run the pipeline on a schedule or based on events, enabling continuous and near real-time synchronization of data from GitHub to ADLS Gen2 without manual intervention.





Implementation in the Project

This setup automates the ingestion of the U.S. House Sales dataset (or any other relevant files) from a centralized GitHub repository directly into the Azure Data Lake. By integrating validation steps and variable controls, the pipeline ensures that only complete and correct data is ingested, improving the reliability and accuracy of downstream analytics and machine learning workflows. The use of triggers facilitates ongoing updates, keeping the data lake synchronized with the latest data commits in GitHub.

6.1 Incremental Ingestion and Bronze Layer Using Databricks AutoLoader

Overview

To enable efficient and continuous ingestion of house sales data into the data lake, we implemented an incremental data loading mechanism using Databricks AutoLoader. This approach minimizes latency, supports schema inference, and enables structured streaming from cloud storage into Delta Lake.

Implementation Details

- **AutoLoader Source Configuration:**
- The raw CSV files are monitored in the raw container of Azure Data Lake Storage Gen2 using the .readStream() method with cloudFiles.format set to csv.
- **Checkpointing:**

A checkpoint directory is configured to allow exactly-once processing and state management, ensuring fault tolerance.

- **Schema Inference:**

AutoLoader uses the schema location to infer and evolve the schema automatically from incoming files.

- **Cleansing Operations:**

The incoming stream is cleaned using PySpark operations:

  - Price and numeric columns are stripped of currency symbols and cast to appropriate types.
  - String cleaning and renaming are done to standardize field names.
  - Null characters and unnecessary columns are removed.
- **Bronze Output:**

The cleaned data stream is continuously written as Delta format to the bronze container for downstream processing.

6.2 Silver Layer Transformation Using Parameterized Notebooks

Overview

In the Silver Layer, we processed and normalized the raw ingested data from the Bronze layer using a parameterized Databricks notebook. This layer standardizes the schema, improves readability, and prepares the data for analytical and ML modeling.

Implementation Details

- **Parameterization via Widgets:**
  - sourcefolder and targetfolder widgets were defined in the notebook.
  - These parameters allow dynamic folder-based ingestion and output without hardcoding paths.
- **Data Loading:**

Bronze data is read from the parameterized sourcefolder using .read().

- **Data Transformation:**

Column names are standardized using .withColumnRenamed() to maintain consistent naming conventions across layers.

- **Silver Output:**

Data is saved in Delta format to the silver container in ADLS Gen2 using .write.format("delta").

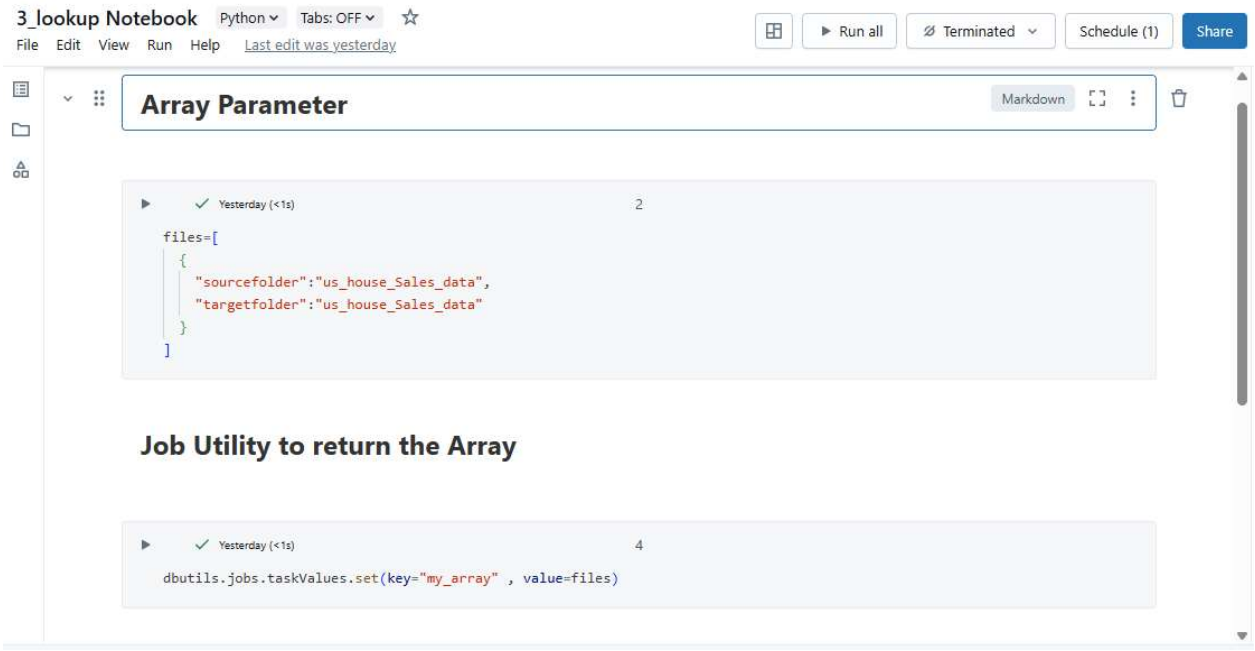
6.3 Job Orchestration with Task Iteration

Overview

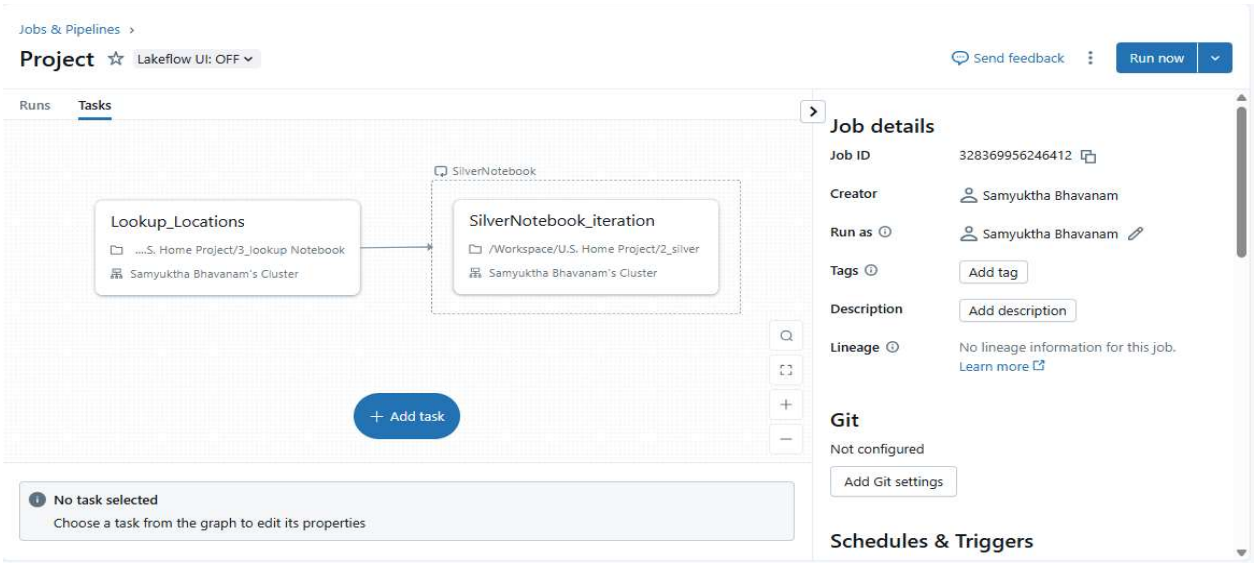
To automate and modularize the Silver layer transformation for multiple datasets or folders, we implemented a Databricks Job that iterates over an array of folder configurations and executes the transformation notebook accordingly.

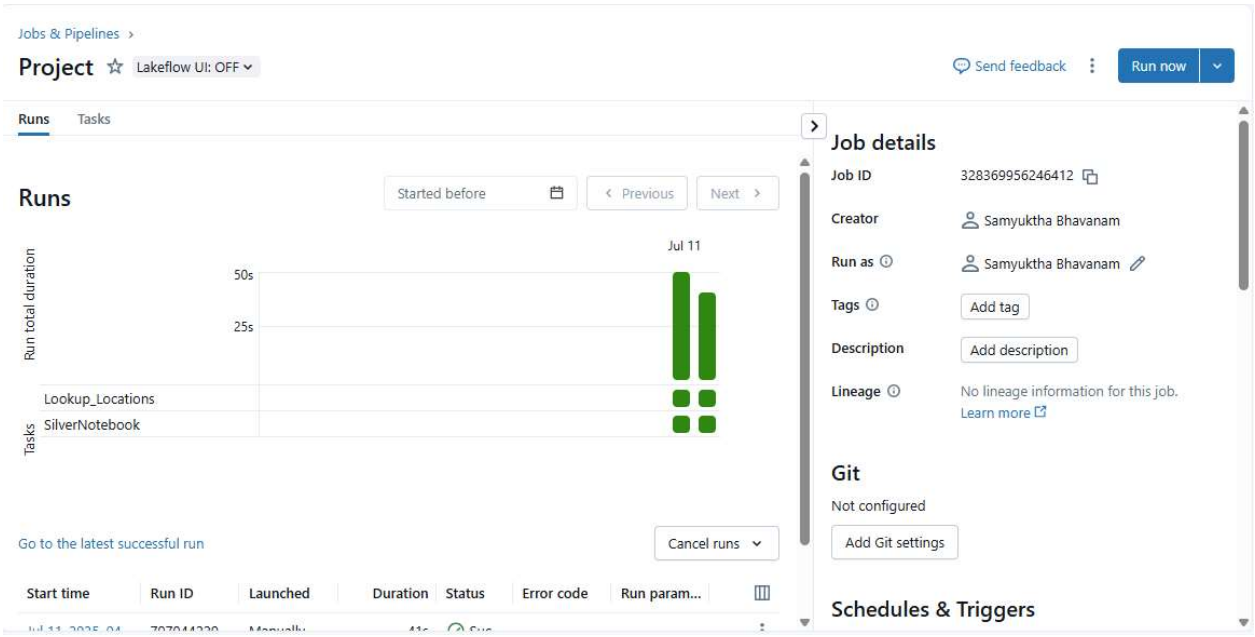
Implementation Steps

- **Notebook 3: Job Control Script**
  - A Python list (files) was defined to hold dictionary entries specifying sourcefolder and targetfolder.
  - The array is passed to downstream tasks using:



- **Iteration Execution:**
  - This array is looped over by the Databricks Job, and for each pair, the Silver Notebook is executed with the corresponding sourcefolder and targetfolder values.
  - This enables dynamic processing of multiple datasets using a single reusable notebook.





7 Gold Layer Transformation and Conditional Job Execution

7.1 Gold Layer Transformation with Price Ranking

Overview

The final layer of the ETL pipeline involves enriching the cleaned Silver data with business logic, such as ranking properties based on sale price. This transformation is saved in the Gold Layer for analytical and reporting use cases.

Implementation Details

- **Source Data:**

Data is read from the Bronze-derived Silver layer (ushome_sales) stored in Delta format.

- **Data Casting:**

Numeric columns such as Price, Bedrooms, Bathrooms, etc., are explicitly cast to appropriate types to ensure consistency.

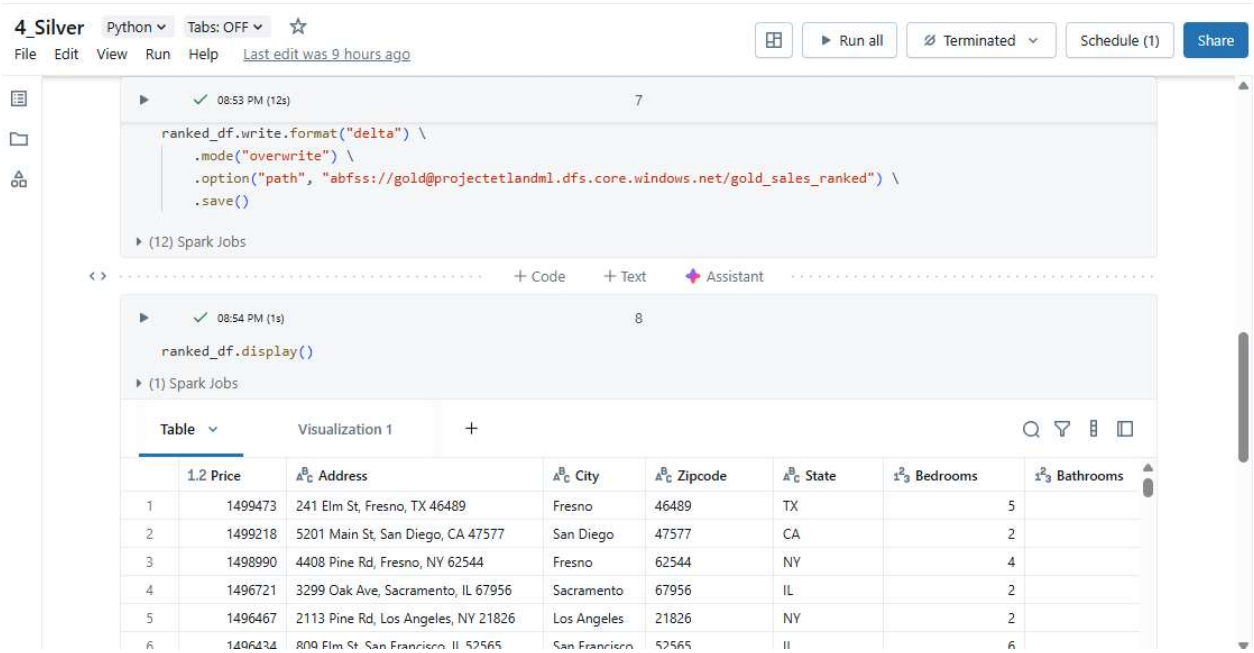
- **Ranking Logic:**

The dataset is enhanced with a new column Price_ranking, using PySpark's dense_rank() window function, ordered by descending price. This allows quick identification of the most valuable listings.

```
08:53 PM (<1s)                                                    6
ranked_df = silver_df.withColumn("Price_ranking", dense_rank().over(Window.orderBy(col("Price").desc())))
ranked_df:  pyspark.sql.dataframe.DataFrame = [Price: double, Address: string ... 16 more fields]
```

- **Gold Output:**

The final ranked dataset is written to the gold container in ADLS Gen2 using Delta format



	1.2 Price	Address	City	Zipcode	State	Bedrooms	Bathrooms
1	1499473	241 Elm St, Fresno, TX 46489	Fresno	46489	TX	5	
2	1499218	5201 Main St, San Diego, CA 47577	San Diego	47577	CA	2	
3	1498990	4408 Pine Rd, Fresno, NY 62544	Fresno	62544	NY	4	
4	1496721	3299 Oak Ave, Sacramento, IL 67956	Sacramento	67956	IL	2	
5	1496467	2113 Pine Rd, Los Angeles, NY 21826	Los Angeles	21826	NY	2	
6	1496434	809 Elm St, San Francisco, IL 52565	San Francisco	52565	IL	6	

7.2 Weekday-Based Conditional Execution Using Task Widgets

Overview

To demonstrate conditional branching in the job workflow, a control notebook (Notebook 5) was introduced. It reads a weekday input, and based on the value (e.g., if it's Friday), triggers execution of the Silver notebook dynamically.

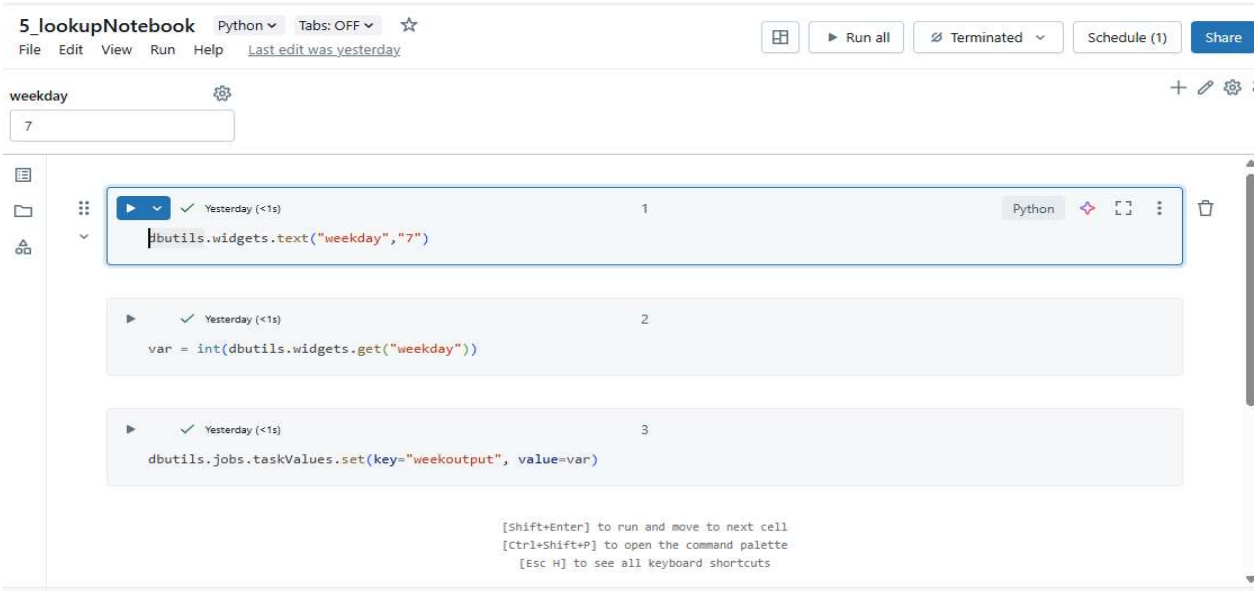
Implementation Steps

- **Parameter Setup:**

A widget weekday is declared and captured via

- **Passing Values:**

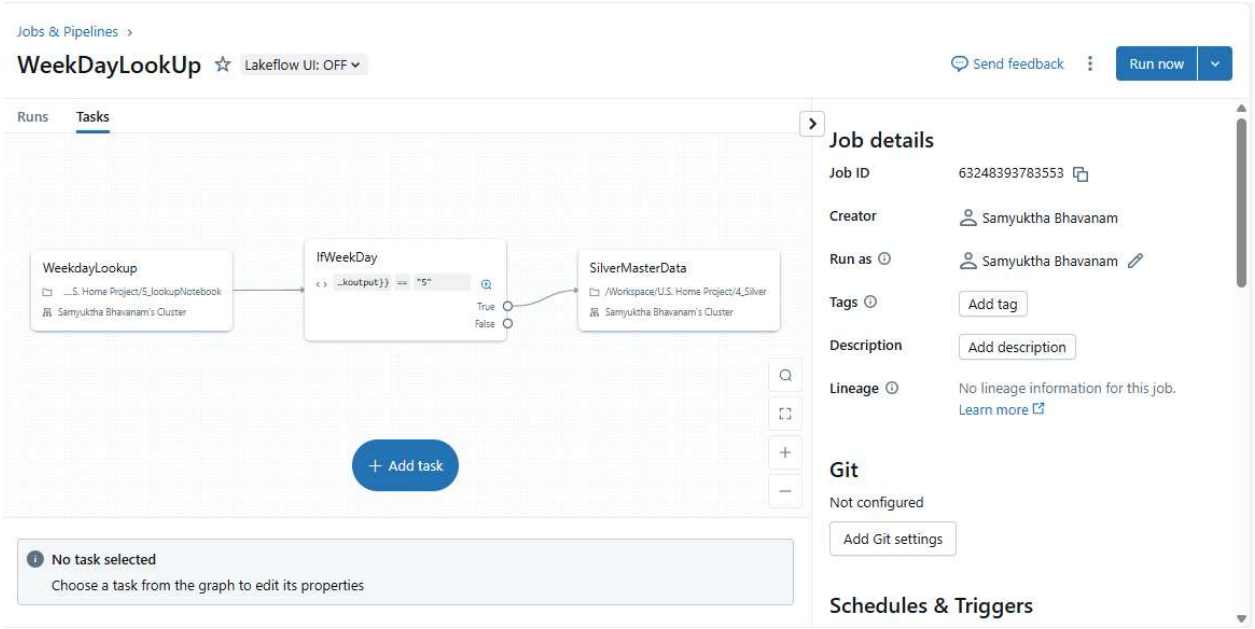
The value is passed to downstream tasks

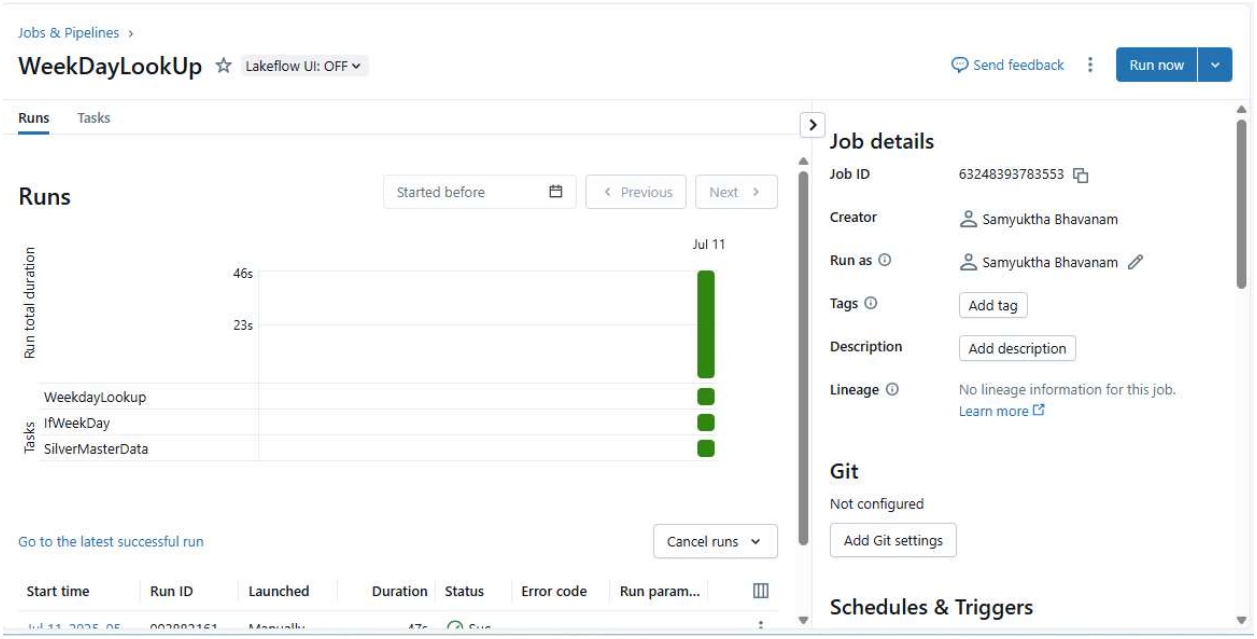


- **Conditional Task Execution in Job:**

In the Databricks Job workflow:

  - Use if-condition logic on the weekoutput task value.
  - Execute the Silver Notebook only if weekoutput == 5 (Friday).
  - This enables dynamic, condition-based orchestration of ETL steps.





8 Delta Live Tables (DLT) Pipeline for Gold Layer

Overview

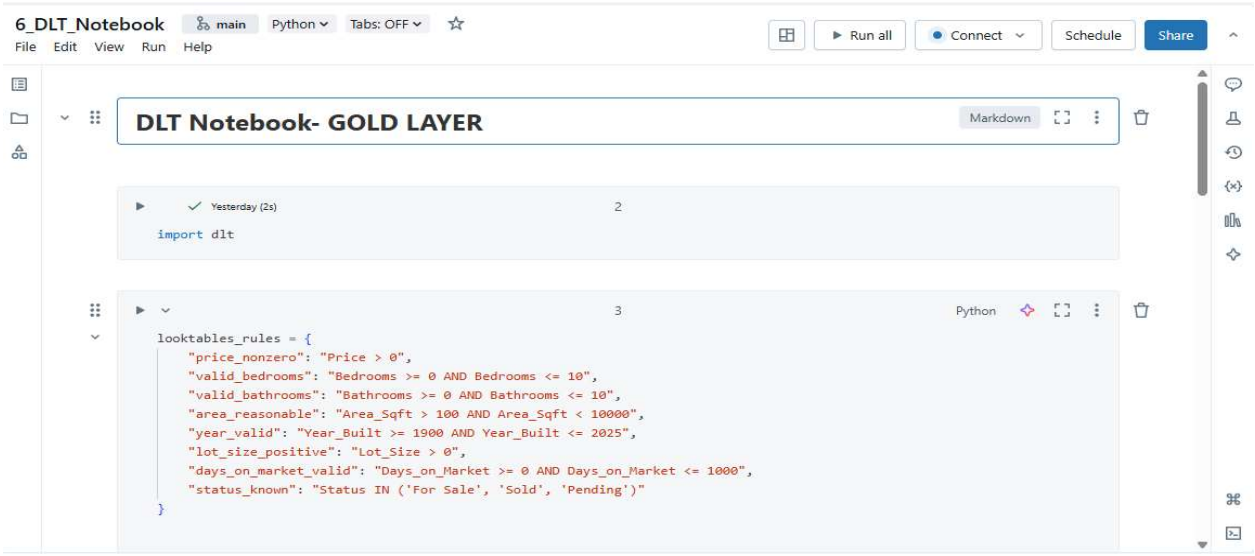
To enhance scalability, observability, and automated quality enforcement in the final stage of the ETL pipeline, a Delta Live Tables (DLT) pipeline was implemented. This ensures continuous, validated transformation of the curated data into the Gold layer using declarative streaming pipelines.

Key Features of DLT Used:

- Streaming-based transformation
- Layered architecture (staging → transformation → gold output)
- Data quality rules (expectations)
- Schema enforcement and monitoring

8.1 Data Quality Expectations

Custom validation rules (expectations) were defined using @dlt.expect_all_or_drop to ensure data integrity:



These rules were applied to the main Gold table (gold_ushouseprice) using:

@dlt.table(name="gold_ushouseprice")
@dlt.expect_all_or_drop(looktables_rules)

8.2 DLT Table Structure and Pipeline Flow

View Name	Purpose
gold_ushouseprice	Main validated Gold table with all quality filters applied
gold_stg_ushouse	Staging layer loading from raw Silver data (streaming)
gold_trns_ushouse (view)	Transformation view to apply additional business logic (adds newflag)

gold_ushouse	Final curated Gold output with additional master-level validation (rule1)

8.3 Business Transformation Example

In the gold_trns_ushouse view, additional logic was applied:

```
7
@dlt.view
def gold_trns_ushouse():
    df = spark.readStream.table("LIVE.gold_stg_ushouse")
    df = df.withColumn("newflag", lit(1))
    return df
```

This transformation helps in tagging or partitioning downstream layers for further use cases like ML modeling or analytics.

8.4 Additional Master Data Rules

Further constraints were applied at the final output stage:

```
8
Python
masterdata_rules={
    "rule1":"newflag is NOT NULL",
}
```

This guarantees that only processed and enriched data is pushed to the final Gold output (gold_ushouse).

9 Machine Learning Pipeline: House Price Prediction & Anomaly Detection

9.1 ML Objective

The goal of this phase is to build a regression model to predict house sale prices based on features such as size, age, and number of rooms, and also detect anomalous property records using an unsupervised approach.

9.2 Data Preparation

- Source data was loaded from the Gold Layer Delta table
- The following input features were selected:

    - Bedrooms, Bathrooms, Area_Sqft, Lot_Size, Year_Built, Days_on_Market
- The target column (Price) was renamed as label.
- Null values were dropped, and the dataset was split into 80% training and 20% testing.

9.3 Linear Regression Model

- **Feature Engineering:**

Used VectorAssembler to assemble features into a single vector column.

- **Model Training:**

Trained a LinearRegression model using Spark ML Pipeline.

- **Model Evaluation:**
    - RMSE was computed using RegressionEvaluator.
    - Relative RMSE was calculated as a percentage of the mean sale price.
- **Model Saving & Output:**
    - Saved the model to: dbfs:/models/house_price_model
    - Predictions were written to Delta in gold layer as predicted_sales

9.4 Anomaly Detection Using Isolation Forest (Sklearn)

- Converted Spark DataFrame to Pandas for scikit-learn compatibility.
- Used IsolationForest with a contamination rate of 1% to identify outlier records.
- Detected records with anomaly == -1 were filtered and converted back to a Spark DataFrame.
- Final anomalies were stored in Delta format in gold layer as anomalies.

9.5 Random Forest Regressor with Log Transformation

- To reduce skew in target values, applied
- Trained a RandomForestRegressor using
    - numTrees = 100
    - maxDepth = 10
- Applied expm1() to revert predictions back to the original scale.
- Evaluated using RMSE and Relative RMSE.
- Model saved to: dbfs:/models/house_price_rf
- Predictions saved to gold layer as predicted_sales_rf

10 Version Control with Azure DevOps and Git Integration in Databricks

10.1 Overview

To ensure reproducibility, collaborative development, and facilitate deployment automation, the project leveraged Azure DevOps Git repositories integrated seamlessly with Azure Databricks. This integration enables centralized version control of all notebooks and related artifacts, thereby supporting best practices in data engineering and ML operations (MLOps).

10.2 Azure DevOps Repository Setup

- A dedicated Git repository was created within Azure DevOps Repos under the project's organizational workspace.
- The repository was named projectetlandml
- To maintain logical structure and ease of navigation, a dedicated folder was created within the repo: /projectetlandml/U.S.Home Project/

This folder houses all notebooks and scripts related to the U.S. House Sales data pipeline.

10.3 Git Integration within Azure Databricks

Integration Workflow:

1. **Configuring Git Integration:**
    a. Within Azure Databricks, Git integration was enabled through User Settings → Git Integration.
    b. Azure DevOps Services was selected as the Git provider.
    c. Authentication was established using a Personal Access Token (PAT) generated in Azure DevOps.

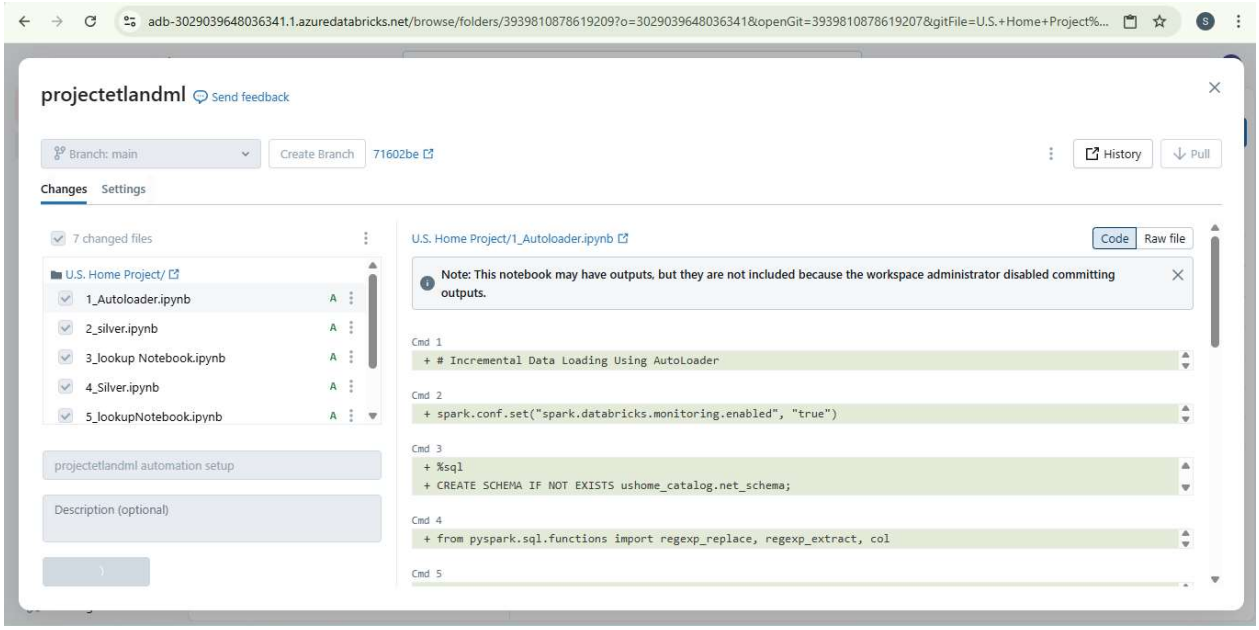
2. **Cloning the Repository:**
    a. From the Repos interface in Databricks, the repository was cloned via Add Repo → Clone Repo.
    b. The repository URL was provided in the format

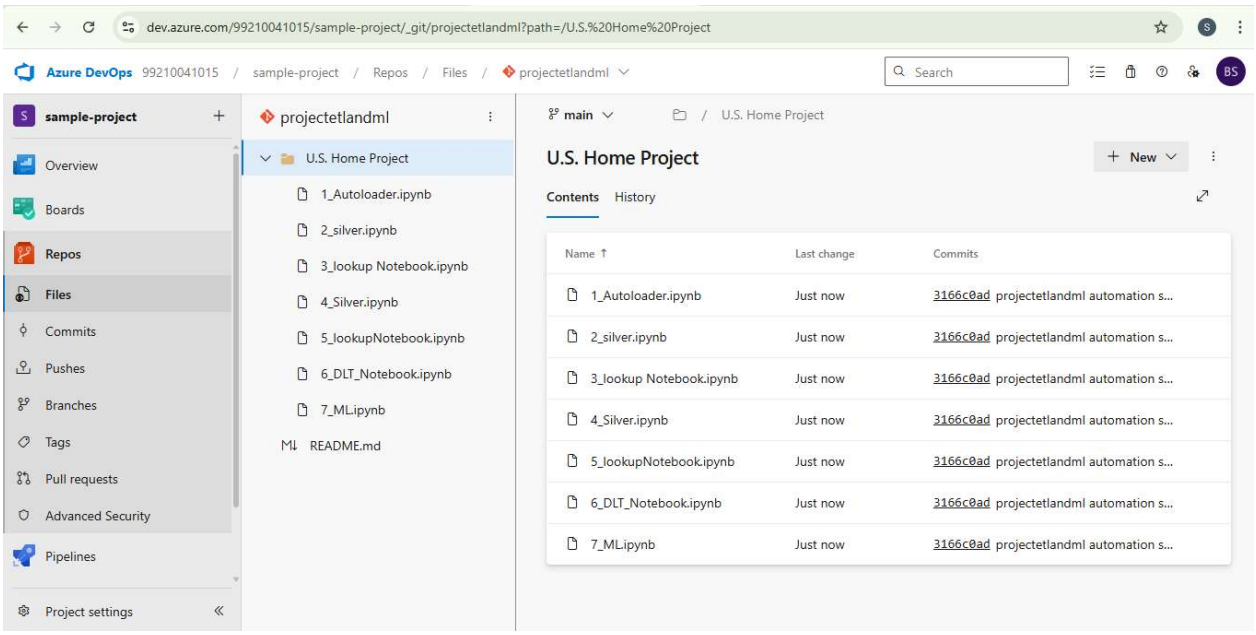
3. **Organizing Project Notebooks:**

a. A new folder structure was established in Databricks under

b. All notebooks pertaining to the project including AutoLoader, Silver and Gold layer processing, job orchestration, Delta Live Tables pipeline, and machine learning model were moved into this folder.

4. **Committing and Pushing Changes:**
   a. Changes were committed locally using the Databricks Git UI.
   b. Commit messages were authored to clearly describe changes.
   c. Commits were pushed upstream to the Azure DevOps remote repository to maintain synchronization.





11 Continuous Integration and Deployment Using Azure DevOps Pipelines

11.1 Overview

To automate the deployment process and enable continuous integration/continuous delivery (CI/CD) for the Databricks notebooks, an Azure DevOps Pipeline was configured. This pipeline pulls the latest code from the Git repository and deploys the notebooks into the Databricks workspace, ensuring consistent and repeatable delivery.

11.2 Pipeline Configuration

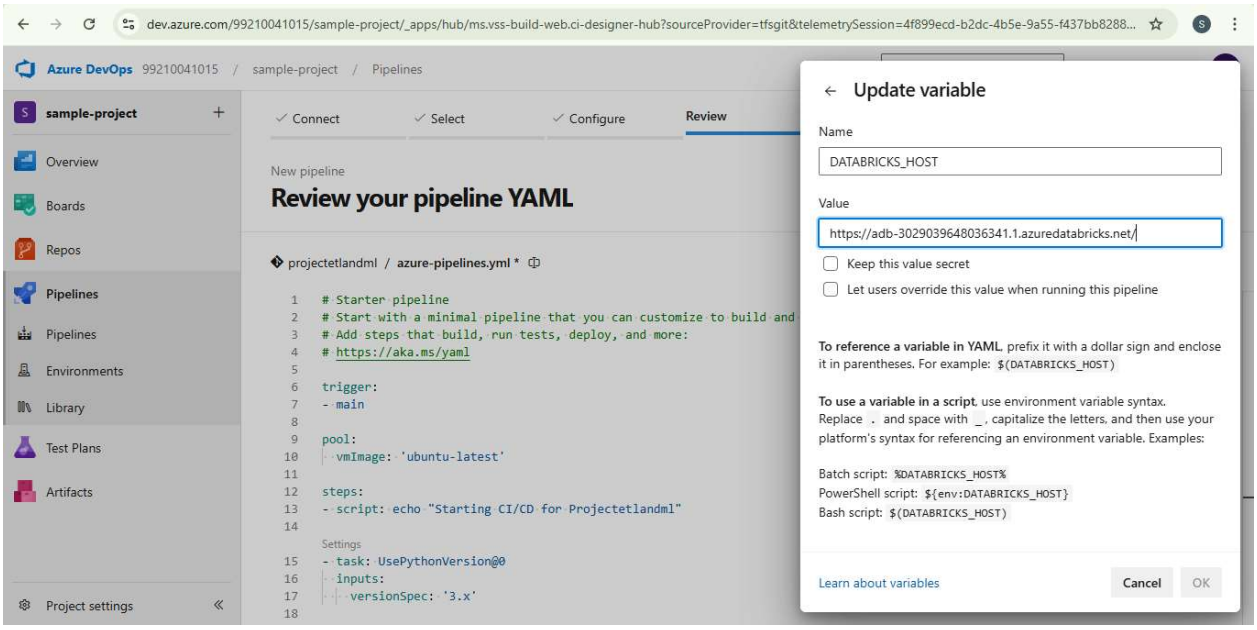
- **Trigger:**
- The pipeline is triggered automatically on every commit to the main branch, enabling continuous integration.
- **Agent Pool:**

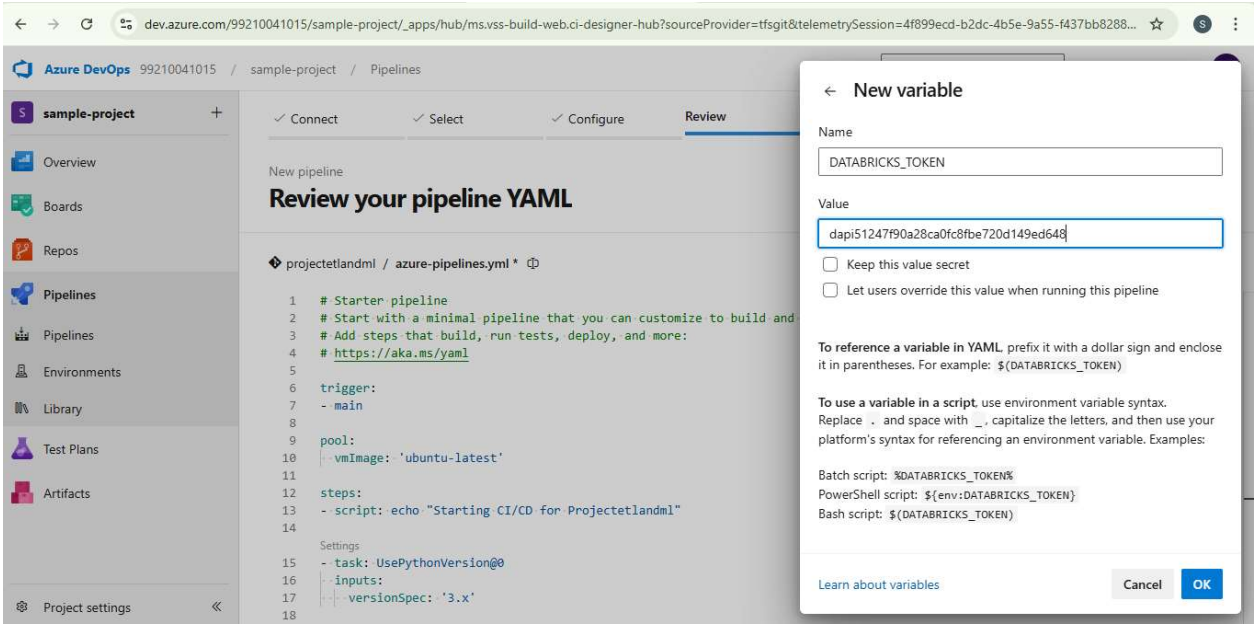
Runs on an ubuntu-latest hosted agent, providing a Linux environment to execute the pipeline tasks.

- **Variables:**

Two key pipeline variables are configured securely in Azure DevOps:

- DATABRICKS_HOST – The Databricks workspace URL.
- DATABRICKS_TOKEN – A personal access token (PAT) for authenticating with Databricks.





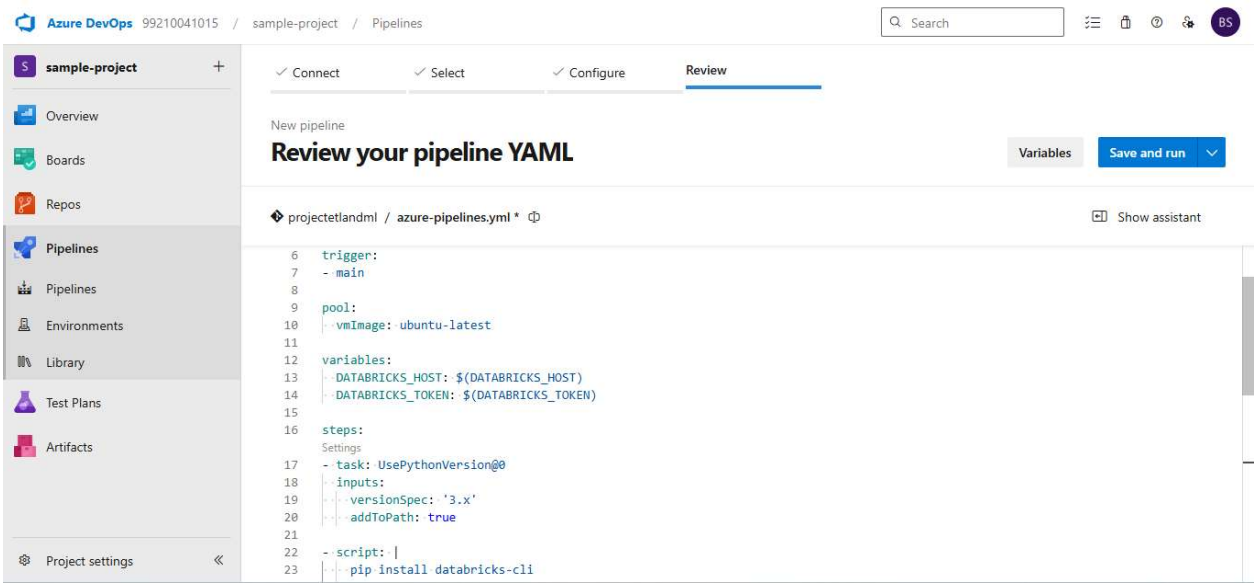
11.3 Pipeline Steps

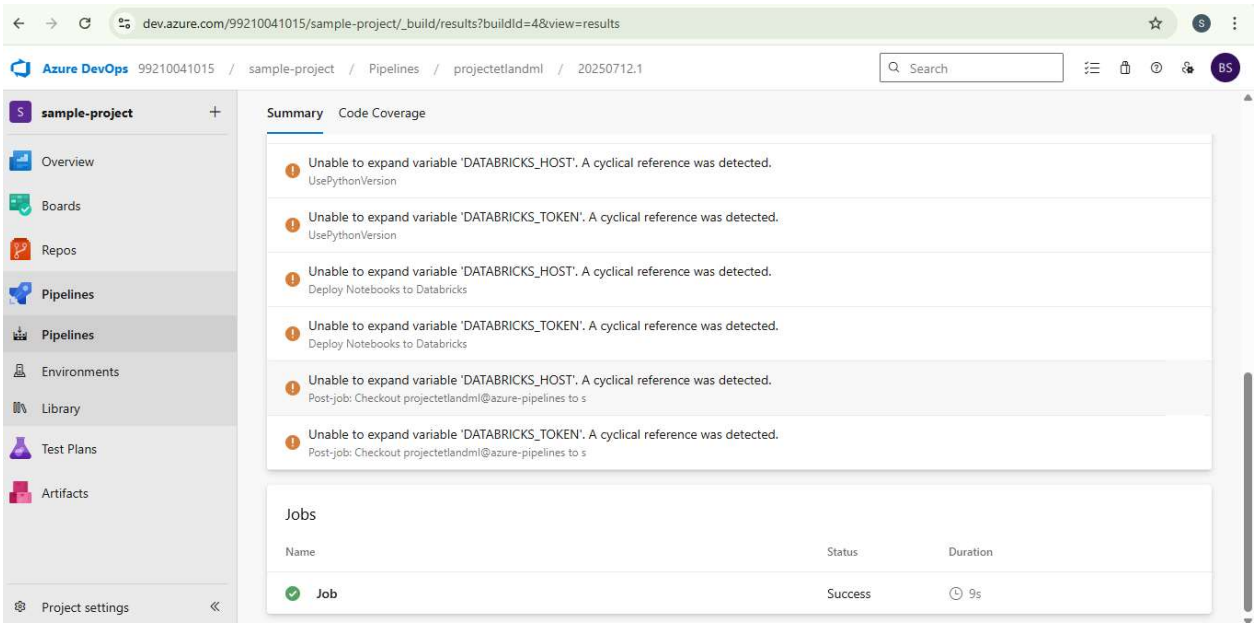
1. **Python Setup:**

The pipeline installs and uses Python 3.x to run scripts and install necessary packages.

2. **Databricks CLI Installation and Configuration:**
    a. Installs the databricks-cli Python package.
    b. Configures the CLI with the Databricks host and token to authenticate API requests.

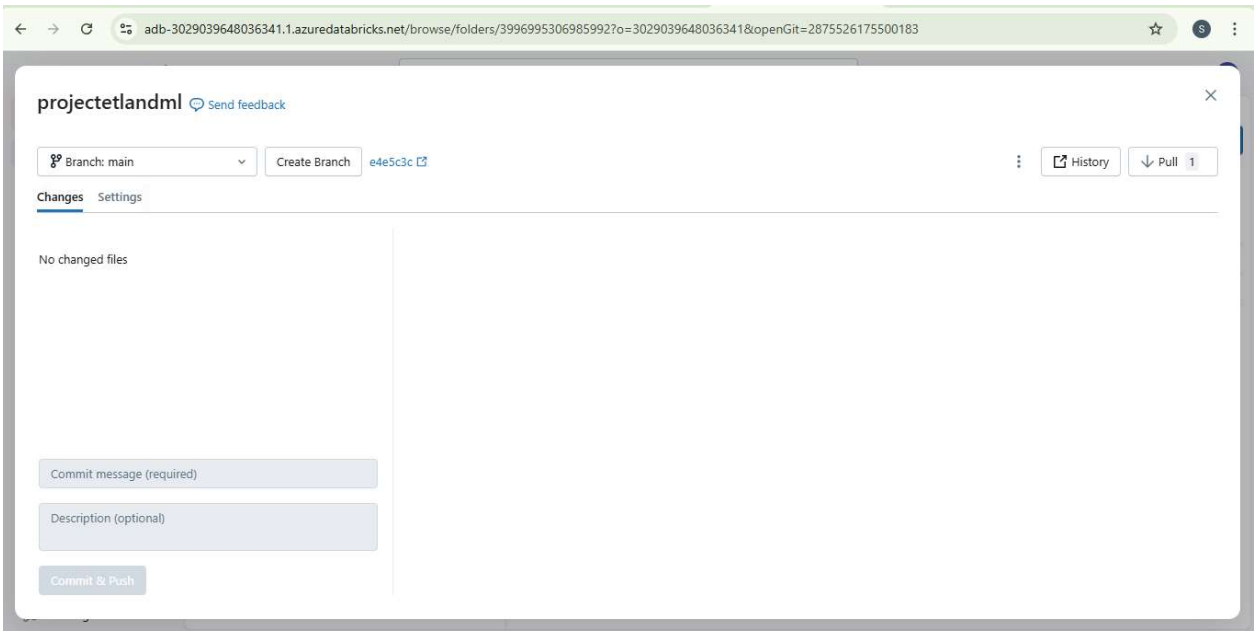
3. **Notebook Deployment Script:**
    a. Recursively searches the ./U.S. Home Project folder in the repo for all Python notebook files (*.py).
    b. Imports each notebook to the Databricks workspace under the folder /Shared/houseproject/.
    c. Uses --format SOURCE and --overwrite flags to ensure the latest code is deployed, overwriting previous versions.

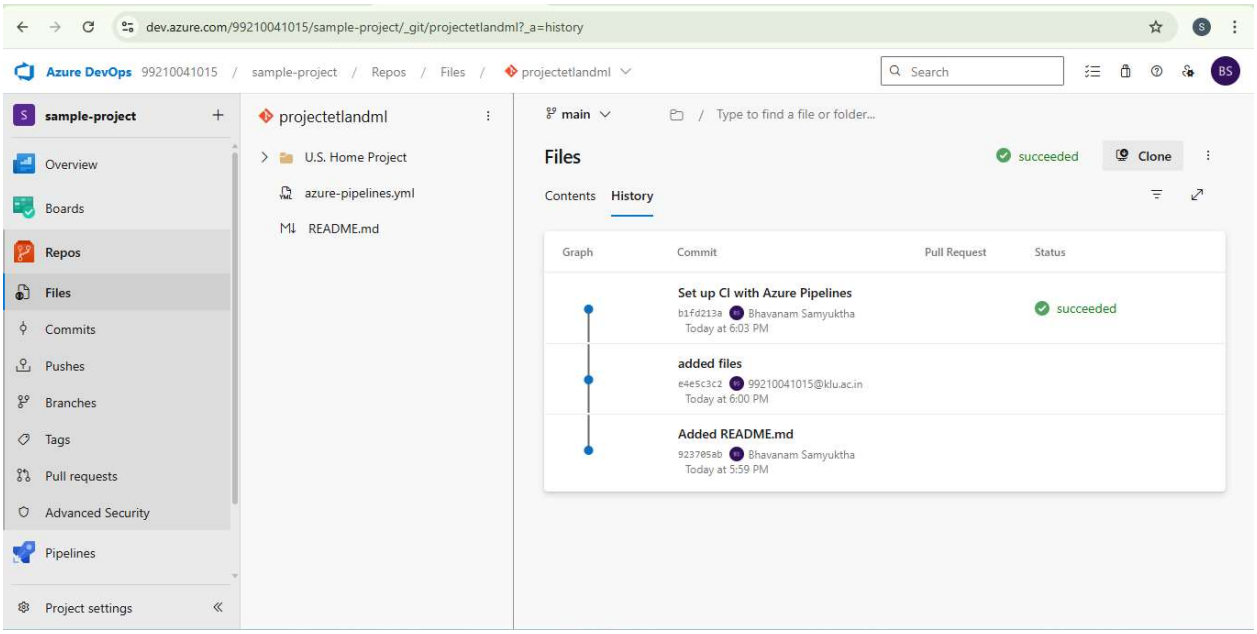




11.4 Pipeline Execution and Outcome

- Upon successful execution, the pipeline displays a success status in Azure DevOps.
- The notebooks are updated in the Databricks workspace repo, reflected by the '1' badge near the Git pull button, confirming the synchronization of the remote Git repo with the Databricks environment.
- This setup ensures that any code changes pushed to the Git repository are automatically deployed to the Databricks workspace, enabling streamlined development and deployment cycles.





12 Monitoring and Logging with Azure Log Analytics

12.1 Use of Azure Log Analytics

Azure Log Analytics is a powerful monitoring and diagnostics service that collects, analyzes, and visualizes telemetry data from cloud resources and applications. In the context of your data engineering and machine learning pipelines, integrating Azure Log Analytics provides:

- **Centralized Monitoring:** Consolidate logs and metrics from Azure Data Factory, Databricks jobs, and other components in one place.
- **Proactive Alerting:** Set up alerts on failures, performance degradation, or anomalies to enable quick response.
- **Performance Insights:** Track pipeline runtimes, data volumes, and error rates to optimize workflows.
- **Audit & Compliance:** Maintain logs for audit trails and compliance reporting.
- **Troubleshooting:** Simplify root cause analysis with detailed logs and diagnostics data.

12.2 Enabling Azure Log Analytics

Step 1: Create a Log Analytics Workspace

- In the Azure Portal, navigate to Log Analytics Workspaces.
- Click Create and specify:
    - Subscription & Resource Group
    - Workspace name (e.g., projectetlandml-logs)
    - Region

Step 2: Enable Diagnostic Settings in Azure Data Factory

- Go to your Azure Data Factory instance.
- Under Monitoring, select Diagnostic settings.
- Add a diagnostic setting to send logs and metrics to your Log Analytics workspace.
- Select categories like PipelineRuns, TriggerRuns, and ActivityRuns.
- Save the settings.

Step 3: Enable Diagnostic Logs for Azure Databricks

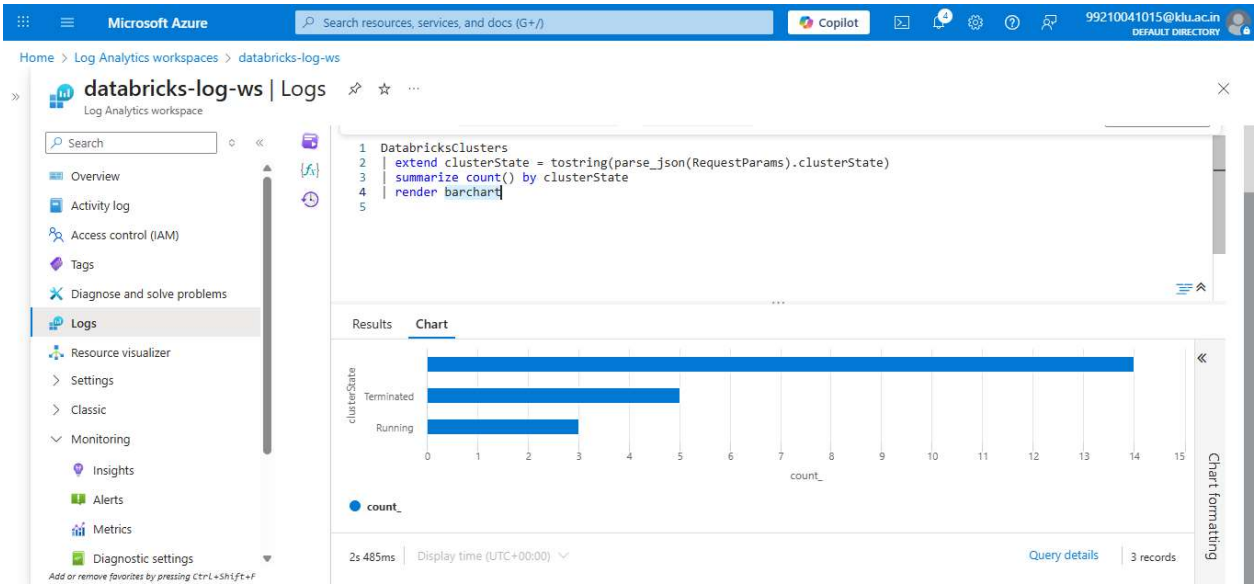
- For enhanced monitoring, enable diagnostic logs for your Databricks workspace via Azure Monitor.
- Forward these logs to your Log Analytics workspace.

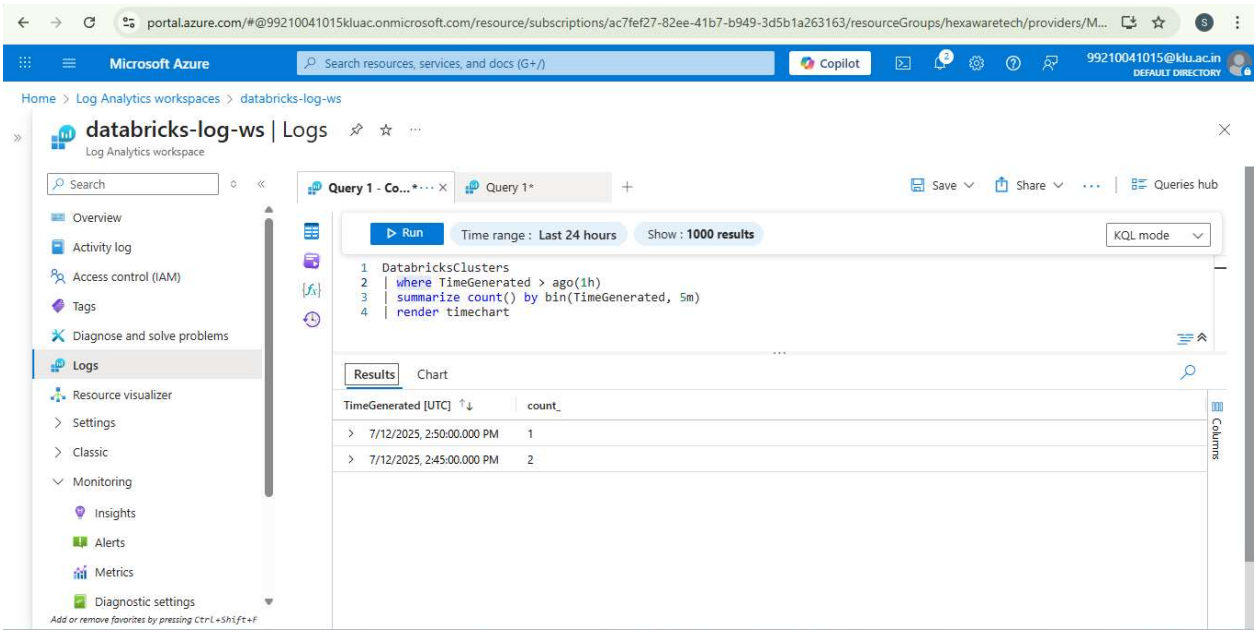
12.3 Using Azure Log Analytics and KQL

Once your logs flow into Azure Log Analytics, you can use Kusto Query Language (KQL) a powerful, intuitive query language to:

- **Query pipeline execution status:** Identify success or failure rates for ETL pipelines, monitor execution times, and filter by specific runs or timeframes.
- **Analyze job performance:** Track the duration and resource usage of Databricks jobs or individual notebook runs.
- **Detect anomalies and failures:** Search for error messages, exceptions, or unusual patterns that indicate potential issues.
- **Aggregate data volumes:** Monitor data ingestion sizes and throughput to detect bottlenecks or spikes.
- **Visualize trends and metrics:** Create dashboards with charts and graphs that provide a real-time overview of your workflows' health.
- **Set alert rules:** Define conditions (e.g., failure count threshold, slow job duration) that trigger automated notifications to your team.

Using KQL enables you to drill down into logs with precision, combine multiple data sources, and perform statistical analysis on your telemetry data—all crucial for maintaining reliable and efficient pipelines.




portal.azure.com/#@99210041015kluac.onmicrosoft.com/resource/subscriptions/ac7fef27-82ee-41b7-b949-3d5b1a263163/resourceGroups/hexawaretech/providers/M...
Microsoft Azure
Search resources, services, and docs (G+/)
Copilot
99210041015@klu.ac.in
DEFAULT DIRECTORY
Home > Log Analytics workspaces > databricks-log-ws
databricks-log-ws | Logs
Log Analytics workspace
Search
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Logs
Resource visualizer
Settings
Classic
Monitoring
Insights
Alerts
Metrics
Diagnostic settings
Add or remove favorites by pressing Ctrl+Shift+F
Query 1 - Co...*
Query 1*
Save
Share
Queries hub
Run
Time range : Last 24 hours
Show : 1000 results
KQL mode
1 DatabricksClusters
2 | where TimeGenerated > ago(1h)
3 | summarize count() by bin(TimeGenerated, 5m)
4 | render timechart
Results
Chart
TimeGenerated [UTC]
count_
7/12/2025, 2:50:00.000 PM
1
7/12/2025, 2:45:00.000 PM
2
Columns