

## Executive Summary

This project focuses on building a modern, dynamic data pipeline using the Twitter dataset with Azure Data Stack, demonstrating practical skills and yielding valuable insights.

## Introduction

In this modern data landscape, everything is driven by data. Scalable, automated and analytical End-to-End pipelines are essential for enabling organisations to effectively process raw data into actionable insights that drive business value.

## Data Source

- The Twitter dataset was sourced and generated by web scraping.
- There are two separate files: one containing over 4000 rows for the initial load and another file containing approximately 400 rows for the incremental process.

## Ingestion Layer - Azure Data Factory

- The initial step is to ingest the raw data into the SQL database.
- Then, ADF was utilised to both ingest and orchestrate the pipeline execution.

## Storage Layer - Azure Data Lake Storage Gen 2

- The ingested data is then staged within Azure Data Lake Storage Gen 2.

## Transformation Layer - Databricks

- Data then flows from the Storage layer to Azure Databricks, where it undergoes transformation processing using PySpark.

## Serving Layer

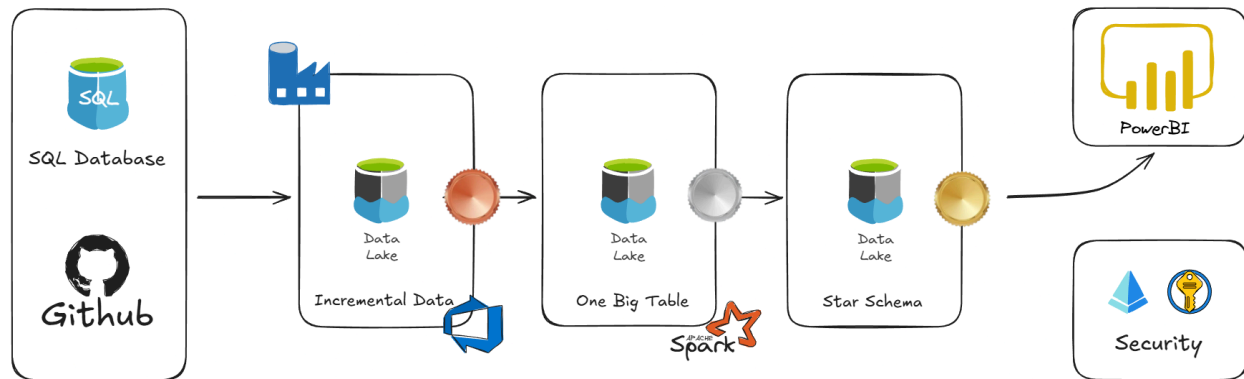
- Following the transformation, the resulting model is served directly.
- Based on the downstream usage, various reporting tools can then be utilised to generate comprehensive reports and dashboards.

## Monitoring

- Implemented using Azure Factory Git CI/CD pipeline.

## Overall Flow Summary

GitHub | SQL Database → ADF → Azure Data Lake Storage Gen 2 → Databricks → Power BI.



Architecture Diagram

The Medallion Architecture, a pattern adopted within Data Engineering, was implemented for this project. It comprises three distinct layers:

- Bronze layer: This layer holds the raw, source-system data.
- Silver layer: This layer stores validated, cleaned and harmonised data
- Gold layer: This layer contains refined, aggregated and feature-engineered data.

## Architecture Overview

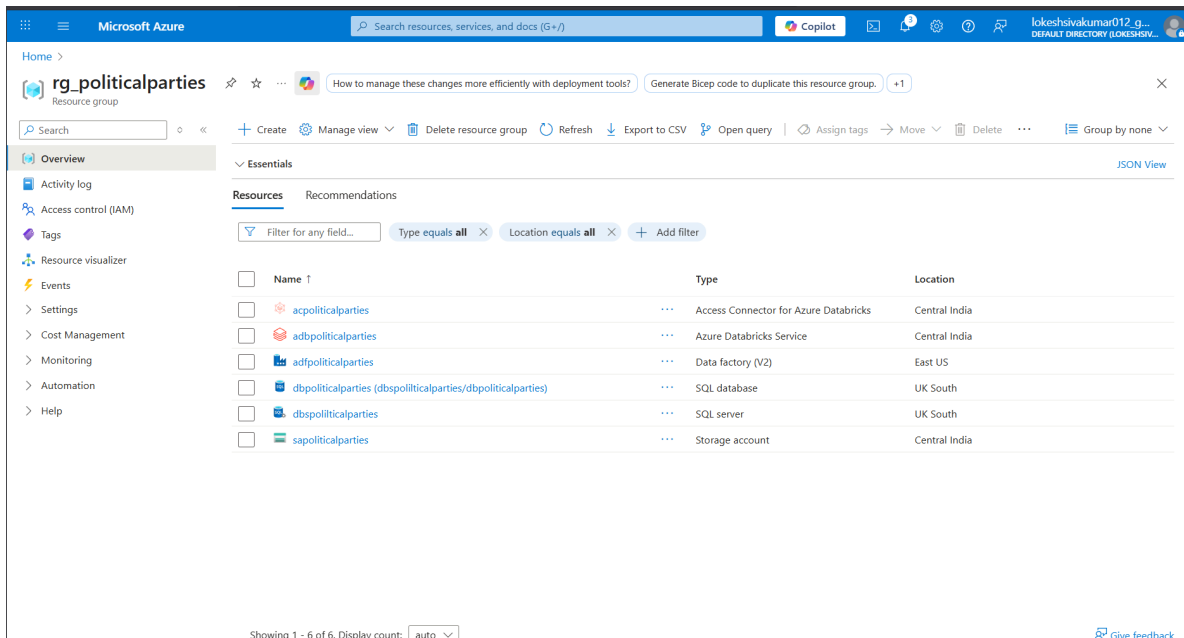
The following components facilitate the pipeline:

- GitHub: Used to store and manage the raw source data.
- SQL Databases: The raw data is initially copied into the local SQL database for staging.
- Azure Data Factory: ADF is used to ingest data into the bronze layer.
- Azure Data Lake Storage Gen 2: This is where all our data gets stored.
- Azure Databricks: Data transformation and refinement occur here, utilising PySpark.
- Azure Key Vault: This service enhances security and secures access credentials.

## Implementation Details

### Environment

Microsoft offers a free 30-day trial period to access its services. I have created the following resource group named `rg_politicalparties`. Within it, I have used the required services, namely Azure Data Factory, Azure Storage Account, Azure Data Lake Storage Gen 2, Azure Databricks, and Azure Databricks Access Connector, as shown in the image below.



## Azure Services

### Data Source

We needed a place to hold our raw data before starting the pipeline. I chose a SQL database for this step. From here, our data is ingested into our bronze layer. So I created a username and password for secure access to our SQL database.

### Data Ingestion Using Azure Data Factory

#### Connecting SQL Server and Azure Data Factory

ADF needed a way to talk to our SQL Server database. They didn't have a connection yet. To fix this, I created a Linked Service to act as a secure bridge between ADF and our SQL database, along with the Dataset so that ADF know exactly which table to read.

### Data Ingestion

I have used the following activities in the ADF to build a flexible pipeline with a parameterisation approach.

- LookUp: To find out the last time we ran the pipeline.
- Set variable: To get the current date using the utcnow function.
- Copy data: From SQL into the Bronze layer.
- If Condition: To check if new data was read, if then
- Script: To get the max value of the last updated cdc column (date), and
- Copy data: To update the last updated date from ingestion.
- Delete: To remove the temporary files.

## Tracking New Data

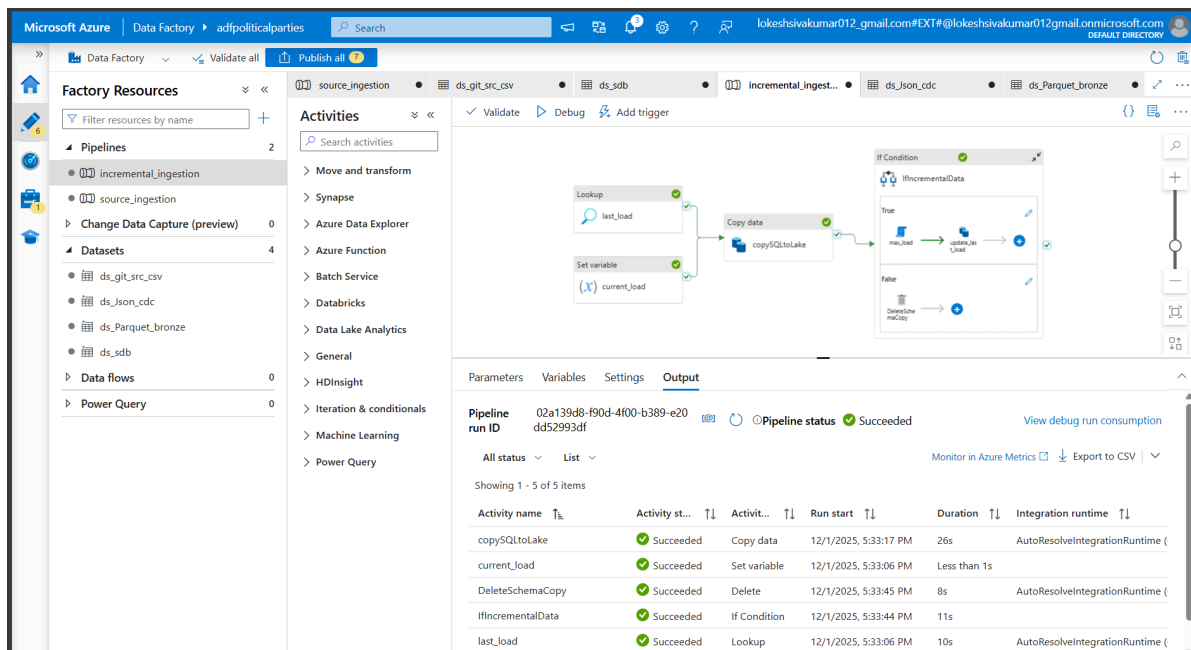
We need a way to remember the last time we loaded data. I also added a back date function. This allows us to re-run data from a past day if needed. I used two files for this:

1. `cdc.json`: Stores the date of the last successful data load.
2. `empty.json`: Helps in interchange of the last ingested date.
3. `back_date`: allows us to rerun old data if needed.

The above parameters help our loop to work effectively.

## Scheduling

After setting up the pipeline, I used the Debug option to run a test. Next, I tested it with the incremental date using an Add Trigger option. Every time it runs, it fetches data from the SQL database. This keeps our Bronze data fresh.



Activity name	Activity st...	Activit...	Run start	Duration	Integration runtime
copySQLtoLake	✓ Succeeded	Copy data	12/1/2025, 5:33:17 PM	26s	AutoResolveIntegrationRuntime (
current_load	✓ Succeeded	Set variable	12/1/2025, 5:33:06 PM	Less than 1s	
DeleteSchemaCopy	✓ Succeeded	Delete	12/1/2025, 5:33:45 PM	8s	AutoResolveIntegrationRuntime (
IfIncrementalData	✓ Succeeded	If Condition	12/1/2025, 5:33:44 PM	11s	
last_load	✓ Succeeded	Lookup	12/1/2025, 5:33:06 PM	10s	AutoResolveIntegrationRuntime (

Incremental Ingestion Loop Pipeline

The raw data gets successfully moved from the SQL database into our Bronze layer. The raw data is stored in one big table: `politicalparties`.

## Deployment and Monitoring

The ADF environment was integrated with a Git repository to track all changes. Any change merged into the main branch automatically triggers the build process, which automates the promotion of the validated pipeline from the development to testing and production.

The screenshot displays the Azure DevOps web interface. The top section shows a pull request titled 'wait' for the 'de\_politicalparties' repository. The pull request is active and has been approved by 'lokesh sivakumar'. A 'Complete pull request' dialog box is open, showing the merge type 'Merge (no fast forward)' and post-completion options. The bottom section shows the 'Files' view of the 'de\_politicalparties' repository, listing files such as 'dataset', 'factory', 'linkedService', 'pipeline', 'cdcjson', 'empty.json', 'IncrementalPolitical.csv', 'PoliticalData.csv', 'publish\_config.json', and 'README.md'.

**Complete pull request**

Merge type: Merge (no fast forward)

Post-completion options:

- ☒ Complete associated work items after merging
- ☒ Delete ls after merging
- ☐ Customize merge commit message

Buttons: Cancel, Complete merge

**Files**

Name	Last change	Commits
dataset	26m ago	d997a4de Adding dataset: ds_json_cdc lokes...
factory	26m ago	d997a4de Adding dataset: ds_json_cdc lokes...
linkedService	26m ago	d997a4de Adding dataset: ds_json_cdc lokes...
pipeline	15m ago	f4f988e6 Renaming pipeline: pipeline1 as wa...
cdcjson	4h ago	5774d5fd Add files via upload Lokesh
empty.json	4h ago	5774d5fd Add files via upload Lokesh
IncrementalPolitical.csv	5h ago	f28b02d8 Add files via upload Lokesh
PoliticalData.csv	5h ago	f28b02d8 Add files via upload Lokesh
publish_config.json	26m ago	7e28f5c0 Update publish_config.json lokes...
README.md	5h ago	c87df01b Initial commit Lokesh

**de\_politicalparties**  
azure data engineer

## CI/CD Pipeline

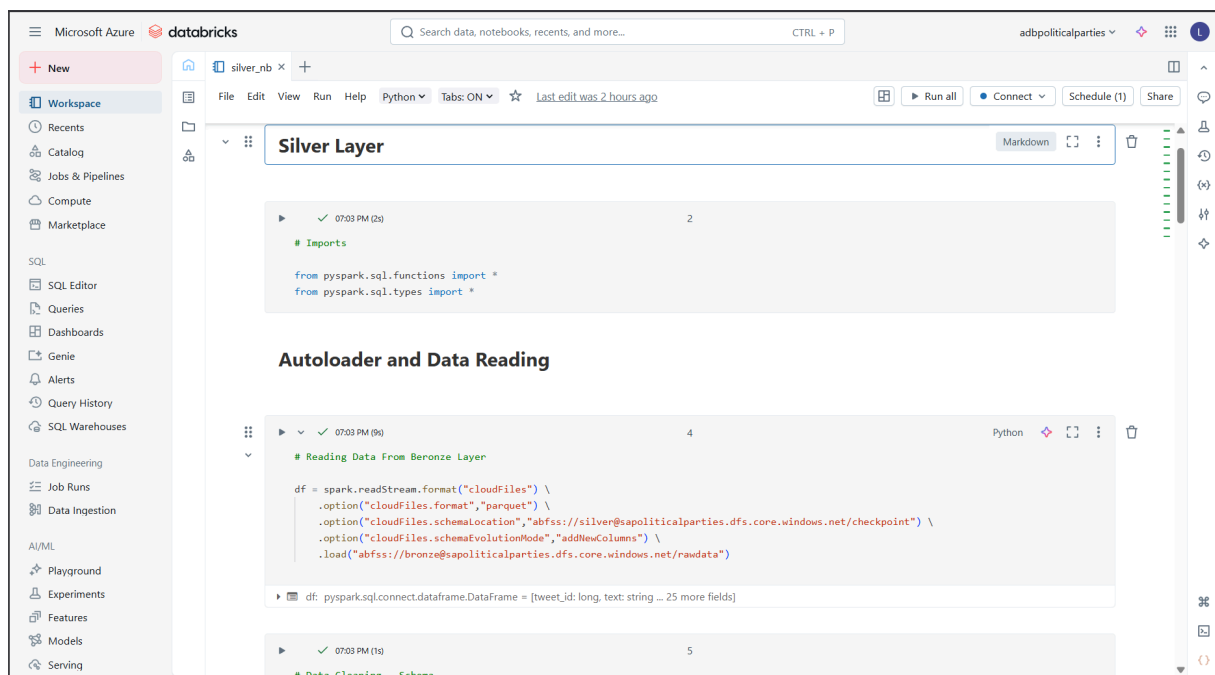
## Data Transformation using Azure Databricks

I have leveraged several tools here:

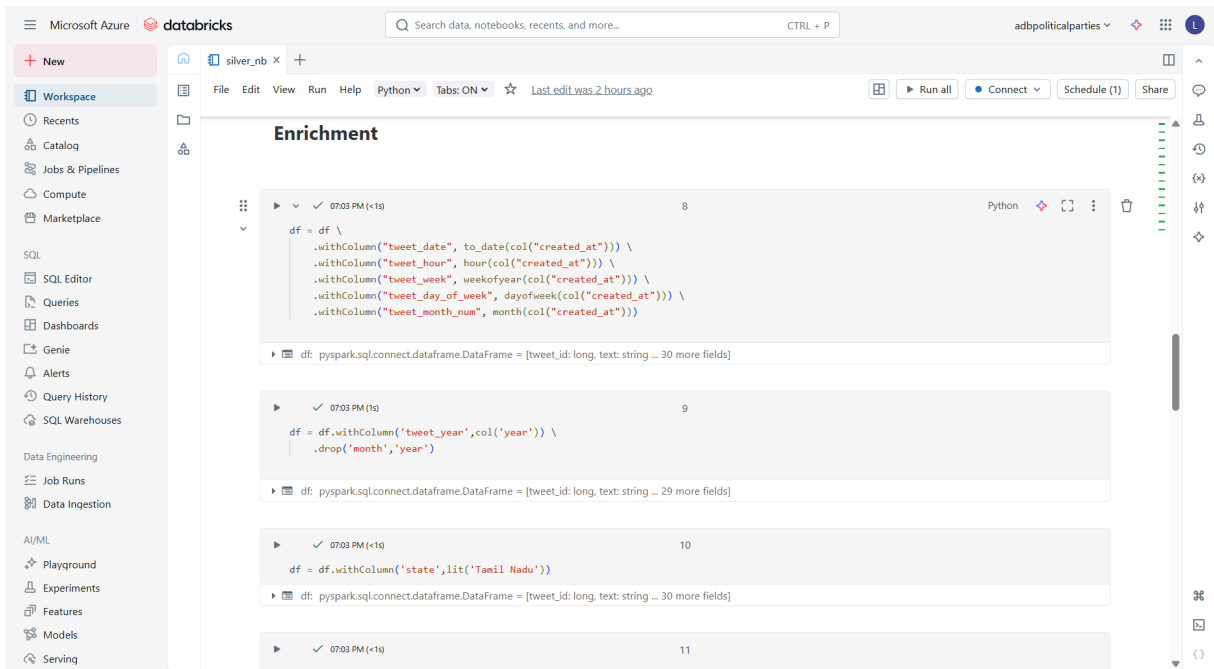
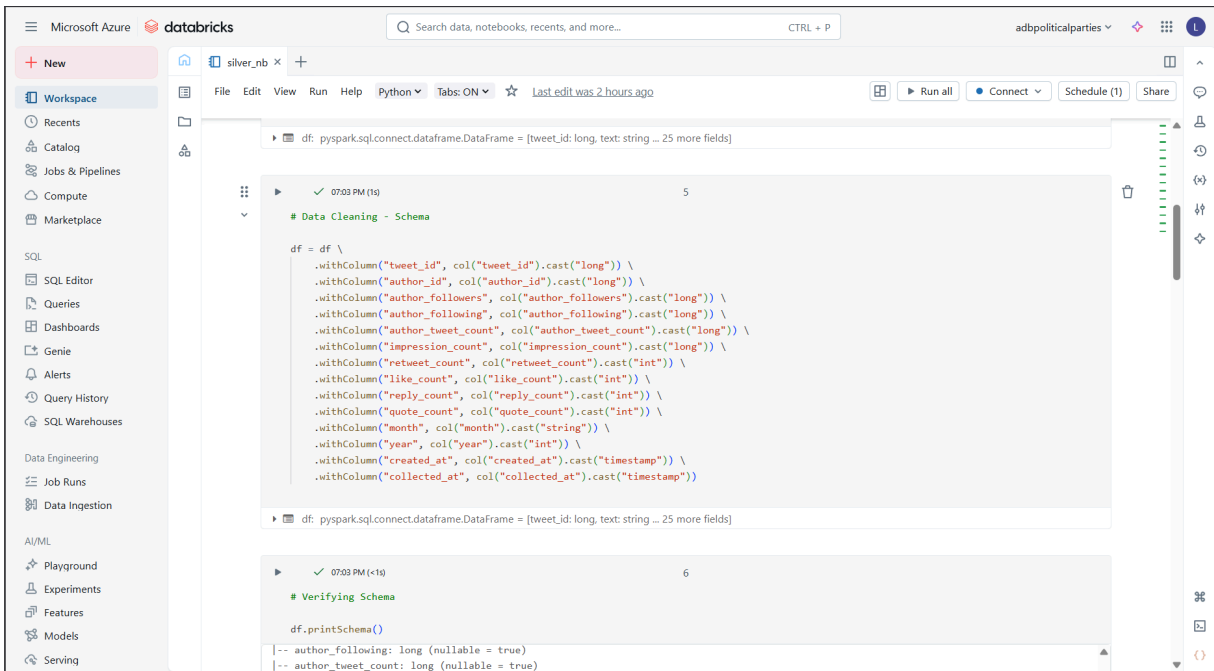
- Databricks Unity Catalog: This helps us manage and secure all our data and tables. It gives us one place to control everything.
- Databricks Access Connector: It uses a Managed Identity, a secure, password-less way for services to connect. It makes our Databricks securely read and write data to ADLS Gen2 without a secret key.
- Spark Streaming with Autoloader: This is the automatic way to handle new data as soon as it arrives. Autoloader watches the files and starts the process automatically.
- Pyspark Transformation: To write all the code for cleaning, validating, and shaping the data.

### The Silver layer

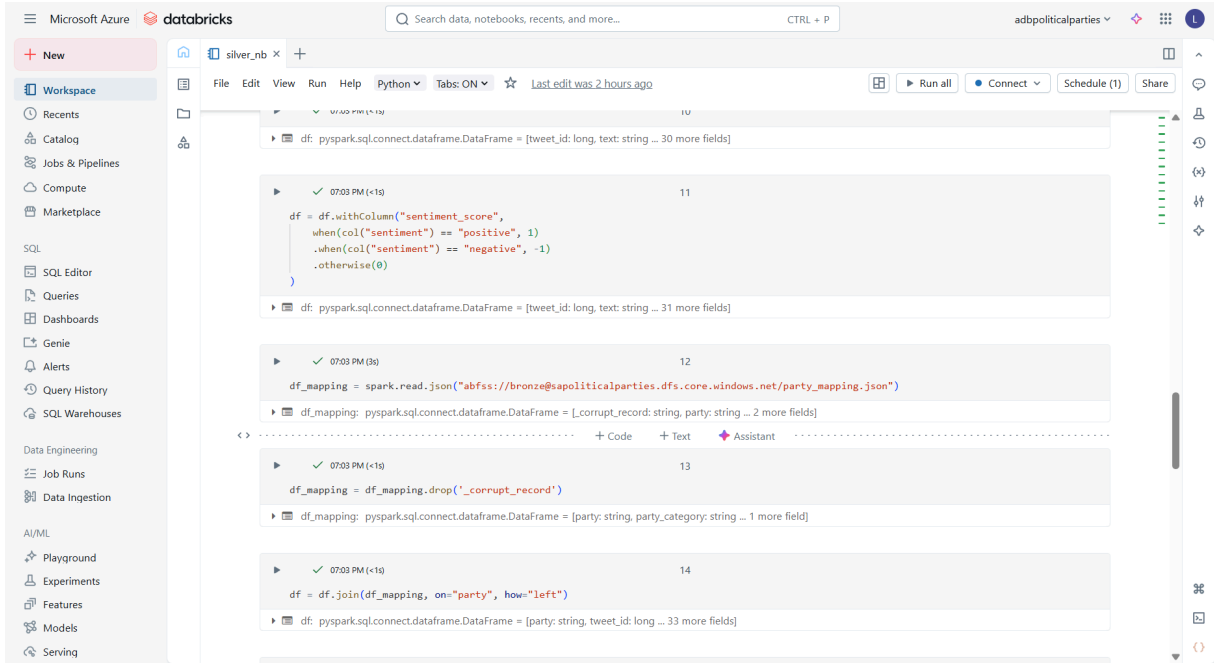
The raw data from the Bronze layer, which we read using the cloudFiles method. This allows us to use streaming for efficiency.



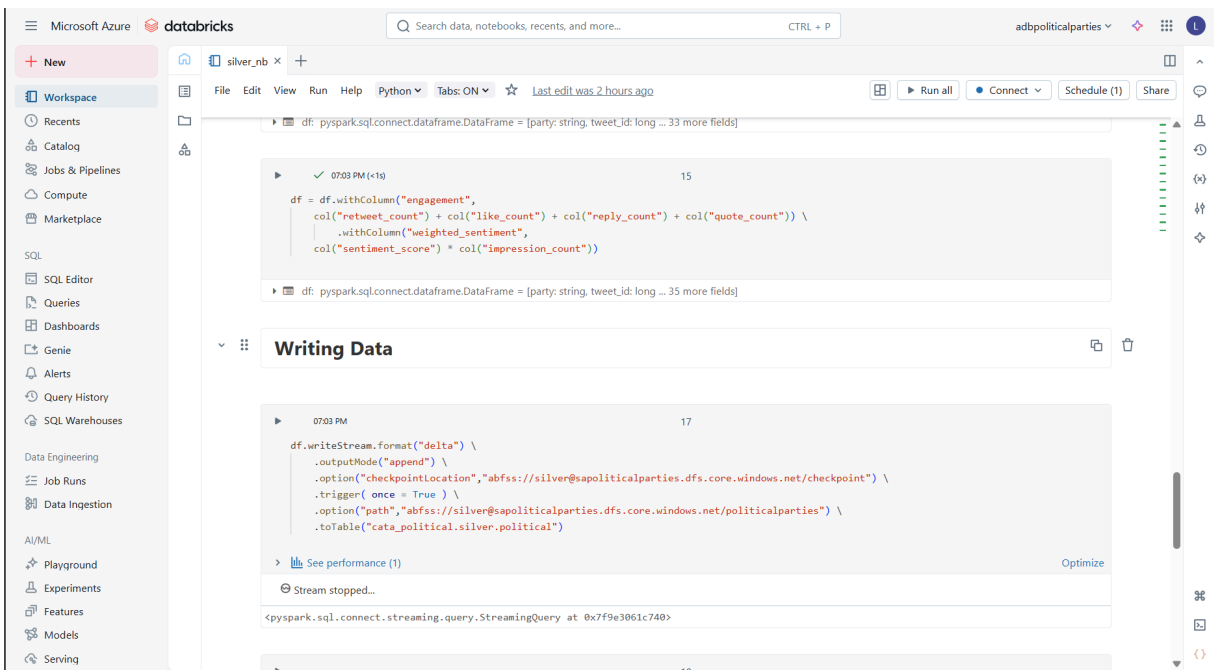
Then I performed a Schema validation, ensuring all the required columns are present and that their names match the defined specification, which includes data type enforcement.



Then, a simple transformation created a few columns with the help of the created\_at column using the withColumn function.



Followed by a mapping process, with the help of a dedicated file that stores the political parties' information alone in a JSON format.

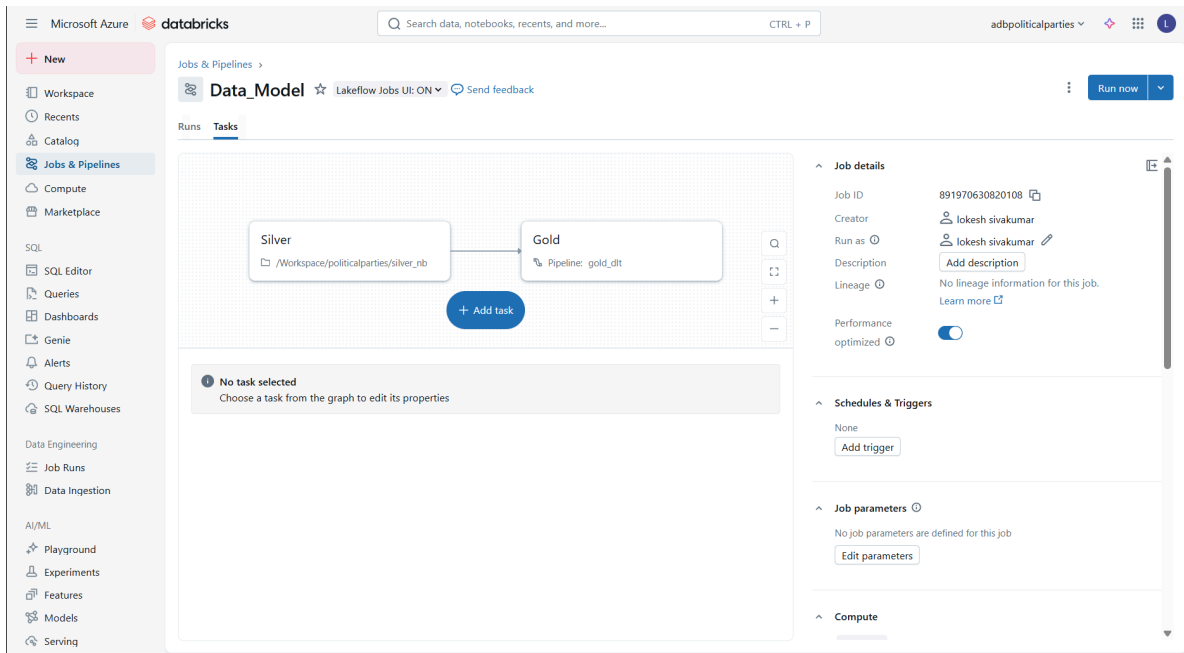


Then I stored it as a Delta Table in the Silver container.



## Gold layer ETL

This layer contains the final optimised data for our downstream purpose.



Gold layer ETL Pipeline

The screenshot shows the Databricks workspace interface for the 'gold\_dlt' pipeline. The pipeline configuration is visible on the left, showing the 'transformations' folder containing files like 'dim\_date.py', 'dim\_district.py', 'dim\_party.py', 'dim\_user.py', 'fact\_tweets.py', and 'silver\_cleaned.py'. The main area displays the 'silver\_cleaned.py' file, which contains the following code:

```
1 import dlt
2 from pyspark.sql.functions import col
3
4 @dlt.table()
5 def silver_cleaned():
6     return (
7         spark.read.format("delta")
8         .load("abfss://silver@apoliticalparties.dfs.core.windows.net/politicalparties")
9         .drop("_rescued_data")
10    )
11
```

Below the code, the 'Tables' section shows a list of tables. The 'SilverCleaned' table is highlighted, showing its details:

Name	Catalog	Sche...	Type	D...	O...	U...	Expec...	D...	W...	F...	D...	Incre...
dim...	cata...	gold	Mater...	4s	3...	-	Not defi	-	-	-	-	Full
dim...	cata...	gold	Mater...	3s	20	-	Not defi	-	-	-	-	Full
dim...	cata...	gold	Strea...	12s	3...	-	Not defi	-	-	-	0	N/A
dim...	cata...	gold	Mater...	3s	3...	-	Not defi	-	-	-	-	Full

SilverCleaned

## Data Modeling

Here, I have transformed the data into a star schema format.

The screenshot shows the Databricks workspace interface. The left sidebar contains navigation options like Workspace, Recents, Catalog, Jobs & Pipelines, Compute, Marketplace, SQL, SQL Editor, Queries, Dashboards, Genie, Alerts, Query History, SQL Warehouses, Data Engineering, Job Runs, Data Ingestion, AI/ML, Playground, Experiments, Features, Models, and Serving. The main area displays the 'gold\_dlt' pipeline configuration. The 'Pipeline assets' section lists 'gold\_dlt' and its sub-items: 'explorations', 'transformations' (containing 'dim\_date.py', 'dim\_district.py', 'dim\_party.py', 'dim\_user.py', 'fact\_tweets.py', 'silver\_cleaned.py'), and 'utilities' (containing 'README.md'). The 'dim\_date.py' file is selected, showing its code in the Lakeflow Pipelines Editor. The code defines a function 'dim\_date()' that returns a table with columns: 'date', 'date\_key', 'day', 'day\_of\_week', 'week', 'month\_num', 'month\_name', 'year', and 'is\_weekend'. The 'Tables' section at the bottom shows a table 'dim\_date' with columns: Name, Catalog, Scheme, Type, D..., O..., U..., Expec..., D..., W..., F..., D..., and Ince....

Name	Catalog	Scheme	Type	D...	O...	U...	Expec...	D...	W...	F...	D...	Ince...
dim_date	cata...	gold	Mater...	4s	3...	-	Not defi	-	-	-	-	Full
dim_date	cata...	gold	Mater...	3s	20	-	Not defi	-	-	-	-	Full
dim_date	cata...	gold	Strea...	12s	-	3...	Not defi	-	-	-	0	N/A
dim_date	cata...	gold	Mater...	3s	3...	-	Not defi	-	-	-	-	Full

DimDate

The screenshot shows the Databricks workspace interface. The left sidebar contains navigation options like Workspace, Recents, Catalog, Jobs & Pipelines, Compute, Marketplace, SQL, SQL Editor, Queries, Dashboards, Genie, Alerts, Query History, SQL Warehouses, Data Engineering, Job Runs, Data Ingestion, AI/ML, Playground, Experiments, Features, Models, and Serving. The main area displays the 'gold\_dlt' pipeline configuration. The 'Pipeline assets' section lists 'gold\_dlt' and its sub-items: 'explorations', 'transformations' (containing 'dim\_date.py', 'dim\_district.py', 'dim\_party.py', 'dim\_user.py', 'fact\_tweets.py', 'silver\_cleaned.py'), and 'utilities' (containing 'README.md'). The 'dim\_district.py' file is selected, showing its code in the Lakeflow Pipelines Editor. The code defines a function 'dim\_district()' that returns a table with columns: 'district'. The 'Tables' section at the bottom shows a table 'dim\_district' with columns: Name, Catalog, Scheme, Type, D..., O..., U..., Expec..., D..., W..., F..., D..., and Ince....

Name	Catalog	Scheme	Type	D...	O...	U...	Expec...	D...	W...	F...	D...	Ince...
dim_district	cata...	gold	Mater...	4s	3...	-	Not defi	-	-	-	-	Full
dim_district	cata...	gold	Mater...	3s	20	-	Not defi	-	-	-	-	Full
dim_district	cata...	gold	Strea...	12s	-	3...	Not defi	-	-	-	0	N/A
dim_district	cata...	gold	Mater...	3s	3...	-	Not defi	-	-	-	-	Full

DimDistrict

The screenshot shows the Databricks workspace interface. The left sidebar contains navigation options like Workspace, Recents, Catalog, Jobs & Pipelines, Compute, Marketplace, SQL, SQL Editor, Queries, Dashboards, Genie, Alerts, Query History, SQL Warehouses, Data Engineering, Job Runs, Data Ingestion, AI/ML, Playground, Experiments, Features, Models, and Serving. The main area displays the 'gold\_dlt' pipeline configuration. The 'Pipeline assets' section lists 'gold\_dlt', 'explorations', 'transformations', 'dim\_date.py', 'dim\_district.py', 'dim\_party.py', 'dim\_user.py', 'fact\_tweets.py', 'silver\_cleaned.py', 'utilities', and 'README.md'. The 'dim\_party.py' file is selected, showing the following code:

```

1 import dlt
2 from pyspark.sql.functions import *
3
4 @dlt.table
5 def dim_party_stg():
6     return (
7         dlt.read("silver_cleaned")
8         .select(
9             col("party").alias("party_name"),
10            "party_code",
11            "party_category",
12            "created_at"
13        )
14        .distinct()
15    )

```

Below the code editor, the 'Tables' tab shows a table with columns: Name, Catalog, Schema, Type, D..., O..., U..., Expec..., D..., W..., F..., D..., and Incre... The table lists several dimension tables, including 'dim\_party'.

## DimParty

The screenshot shows the Databricks workspace interface. The left sidebar contains navigation options like Workspace, Recents, Catalog, Jobs & Pipelines, Compute, Marketplace, SQL, SQL Editor, Queries, Dashboards, Genie, Alerts, Query History, SQL Warehouses, Data Engineering, Job Runs, Data Ingestion, AI/ML, Playground, Experiments, Features, Models, and Serving. The main area displays the 'gold\_dlt' pipeline configuration. The 'Pipeline assets' section lists 'gold\_dlt', 'explorations', 'transformations', 'dim\_date.py', 'dim\_district.py', 'dim\_party.py', 'dim\_user.py', 'fact\_tweets.py', 'silver\_cleaned.py', 'utilities', and 'README.md'. The 'dim\_party.py' file is selected, showing the following code:

```

12 "created_at"
13 )
14 .distinct()
15 )
16
17 dlt.create_streaming_table("dim_party")
18
19 dlt.create_auto_cdc_flow(
20     target = "dim_party",
21     source = "dim_party_stg",
22     keys = ["party_name"],
23     sequence_by = "created_at",
24     stored_as_scd_type = 2,
25     track_history_except_column_list = None,
26     name = None,

```

Below the code editor, the 'Tables' tab shows a table with columns: Name, Catalog, Schema, Type, D..., O..., U..., Expec..., D..., W..., F..., D..., and Incre... The table lists several dimension tables, including 'dim\_party'.

## DimParty SCD

## Star Schema Structure

Fact Table: I have created a central Fact Table containing metrics and foreign keys.

Dimension Table: The data was normalised into several independent Dimension Tables, namely: DimDate, DimDistrict, DimParty and DimUser.

Free trial ends in 14 days. Upgrade to Premium in Azure Portal

Run file Lakeflow Pipelines Editor: ON Last edit was 2 hours ago

```

5 @dlt.table
6 def dim_user_stg():
7     return (
8         dlt.read("silver_cleaned")
9         .select(
10             "author_id",
11             "author_username",
12             "author_name",
13             "author_followers",
14             "author_following",
15             "author_tweet_count",
16             "author_verified",
17             "user_location",
18             "collected_at"
19         )
20     )

```

Name	Catalog	Sche...	Type	D...	O...	U...	Expec...	D...	W...	F...	D...	Incre...
dim_...	cata...	gold	Mater...	4s	3...	-	Not defi	-	-	-	-	Full
dim_...	cata...	gold	Mater...	3s	20	-	Not defi	-	-	-	-	Full
dim_...	cata...	gold	Strea...	12s	-	3...	Not defi	-	-	-	0	N/A
dim_...	cata...	gold	Mater...	3s	3...	-	Not defi	-	-	-	-	Full

Dec 03, 2025, 08:53 PM · 1m 2s · Full refresh all · Query performance

## DimUser

Free trial ends in 14 days. Upgrade to Premium in Azure Portal

Run file Lakeflow Pipelines Editor: ON Last edit was 2 hours ago

```

22
23 dlt.create_streaming_table("dim_user")
24
25 dlt.create_auto_cdc_flow(
26     target="dim_user",
27     source="dim_user_stg",
28     keys=["author_id"],
29     sequence_by="collected_at",
30     stored_as_scd_type = 2,
31     track_history_except_column_list=[
32         "author_followers",
33         "author_following",
34         "author_tweet_count"
35     ],
36     name = None,

```

Name	Catalog	Sche...	Type	D...	O...	U...	Expec...	D...	W...	F...	D...	Incre...
dim_...	cata...	gold	Mater...	4s	3...	-	Not defi	-	-	-	-	Full
dim_...	cata...	gold	Mater...	3s	20	-	Not defi	-	-	-	-	Full
dim_...	cata...	gold	Strea...	12s	-	3...	Not defi	-	-	-	0	N/A
dim_...	cata...	gold	Mater...	3s	3...	-	Not defi	-	-	-	-	Full

Dec 03, 2025, 08:53 PM · 1m 2s · Full refresh all · Query performance

## DimUser SCD

### Slowly Changing Dimensions

It is essential for tracking changes to dimensional attributes over time. This ensures that historical reports reflect the data as it was at the time of the event, providing accurate historical analysis for our downstream team.

The screenshot shows the Databricks Lakeflow Pipelines Editor. The main pane displays the code for the 'fact\_tweets.py' transformation. The code reads from a Delta Live Table named 'silver\_cleaned' and selects various attributes including tweet\_id, tweet\_date, tweet\_hour, tweet\_week, tweet\_day\_of\_week, tweet\_month\_num, tweet\_year, district, party, sentiment, sentiment\_score, engagement, weighted\_sentiment, impression\_count, and author\_id. Below the code, the 'Tables' tab is active, showing a list of tables in the 'cata...' catalog. The tables include 'dim\_date', 'dim\_district', 'dim\_party', and 'fact\_tweets'. The 'fact\_tweets' table is highlighted, showing its type as 'Materialized View' and its refresh status as 'Full'.

## FactTweets

The screenshot shows the Databricks Lakeflow Pipelines Editor. The main pane displays the code for the 'fact\_tweets.py' transformation. The code creates a streaming table named 'fact\_tweets' and sets up an auto-cdc flow. The code includes the following lines:

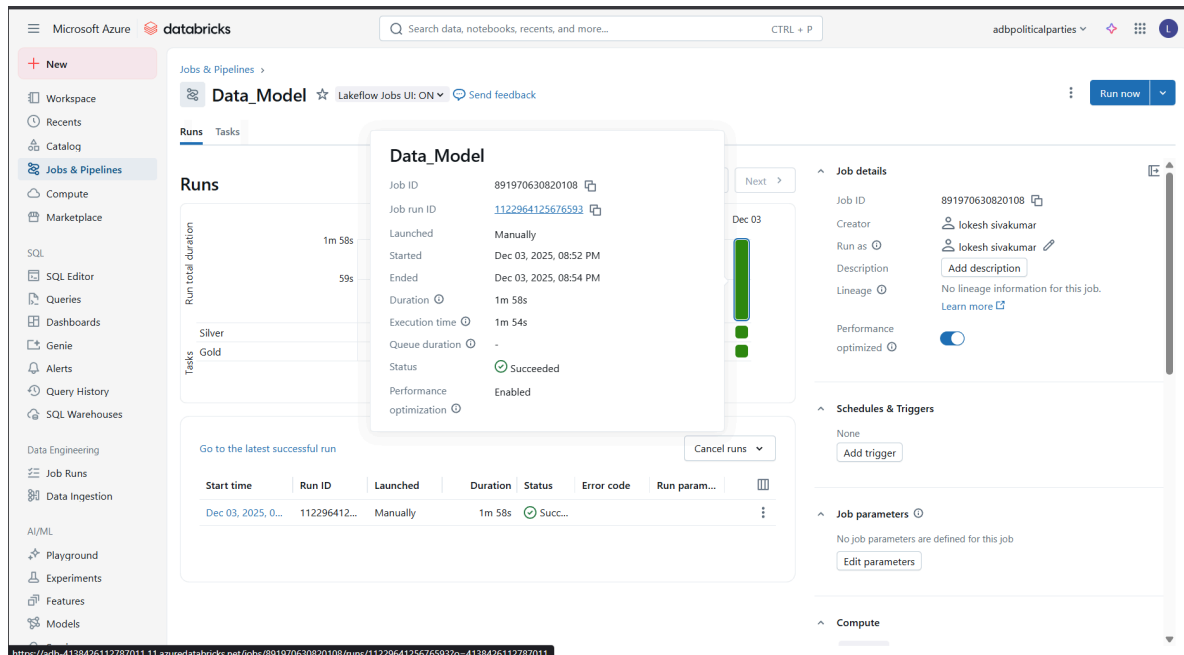
```
dlt.create_streaming_table("fact_tweets")
dlt.create_auto_cdc_flow(
    target="fact_tweets",
    source="fact_tweets_stg",
    keys=["tweet_id"],
    sequence_by="collected_at",
    stored_as_scd_type=1,
    track_history_except_column_list=None,
    name=None,
    once=False
)
```

Below the code, the 'Tables' tab is active, showing a list of tables in the 'cata...' catalog. The tables include 'dim\_date', 'dim\_district', 'dim\_party', and 'fact\_tweets'. The 'fact\_tweets' table is highlighted, showing its type as 'Materialized View' and its refresh status as 'Full'.

## FactTweets SCD

## Delta Live Tables

It is instrumental in managing the complexity of both the star schema creation and the automated implementation of the SCD logic.



Gold Pipeline

## Pipeline Testing

I have tested this pipeline with the incremental data set, and it successfully proved its effectiveness by incorporating the new data without any errors.

## Serving the Data

Once the data reaches the Gold layer, it is ready to be served to the downstream process team, which includes the Data Analyst. With this, I have ensured that our pipeline is industry-level and production-ready.

## Challenges and Solutions

- Web Scraping Limitation → Used rotating headers.
- Data Inconsistency → Standardise formats using Pandas and Pyspark.
- Schema Evolution → Implemented Delta Lake for versioning.

## Conclusion

I built this scalable, cloud-based, enterprise-ready pipeline from scratch, directly facing and solving real-world challenges, and now I am ready to deliver a production-level pipeline with real business insights.