

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

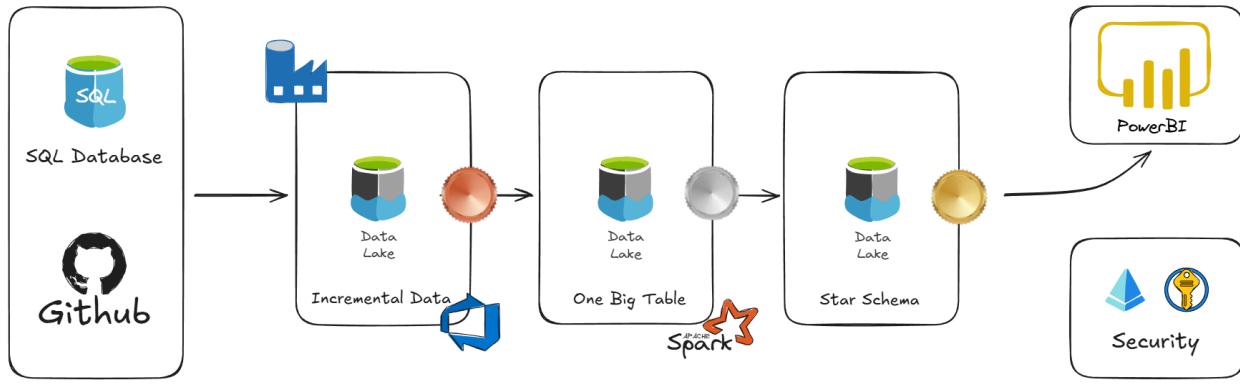
- Following the transformation, the resulting model is served directly.
- Based on the downstream usage, various reporting tools can then be utilised to generate comprehensive reports and dashboards.

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.

Name	Type	Location
acpoliticalparties	Access Connector for Azure Databricks	Central India
adbpoliticalparties	Azure Databricks Service	Central India
adfpoliticalparties	Data factory (V2)	East US
dbpoliticalparties (dbspoliticalparties/dbpoliticalparties)	SQL database	UK South
dbpoliticalparties	SQL server	UK South
sapoliticalparties	Storage account	Central India

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

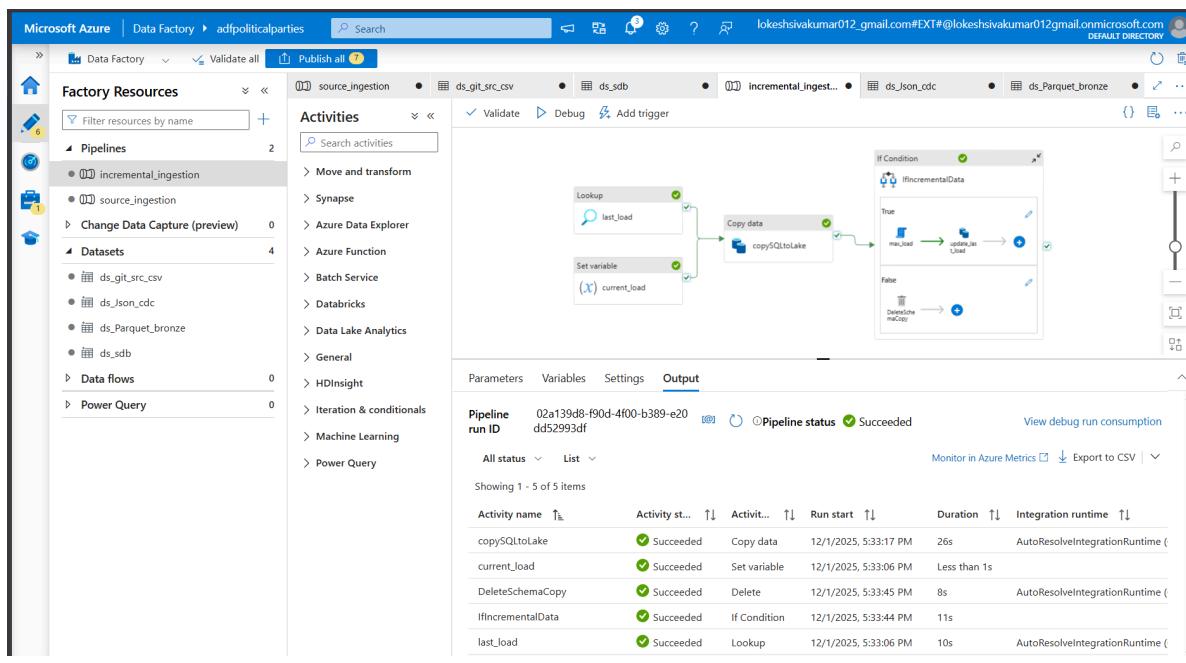
We need a way to remember the last time we loaded data. I also added a back date function. This allows us to re-run data from a past day if needed. I used two files for this:

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.



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 two windows from the Azure DevOps interface.

Top Window: Complete pull request

- Merge type:** Merge (no fast forward) (selected)
- Post-completion options:**
 - Complete associated work items after merging
 - Delete ls after merging
 - Customize merge commit message
- Description:** wait_pipeline
- Comments:** A list of comments from user 'ls' and reviewer 'lokesh sivakumar'.
- Buttons:** Cancel, Complete merge.

Bottom Window: CI/CD Pipeline

- Repository:** de_politicalparties.git
- Branch:** main
- Files:** dataset, factory, linkedService, pipeline, cdc.json, empty.json, IncrementalPolitical.csv, PoliticalData.csv, publish_config.json, README.md
- Build Status:** Set up build
- Clone:** Clone
- Commit History:** A table showing commits for each file.
- Project Information:** 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.

```
# Imports
from pyspark.sql.functions import *
from pyspark.sql.types import *

# Reading Data From Beronze Layer
df = spark.readStream.format("cloudFiles") \
    .option("cloudfiles.format","parquet") \
    .option("cloudfiles.schemaLocation","abfss://silver@sapoliticalparties.dfs.core.windows.net/checkpoint") \
    .option("cloudfiles.schemaEvolutionMode","addNewColumns") \
    .load("abfss://bronze@sapoliticalparties.dfs.core.windows.net/rawdata")

# df: pyspark.sql.connect.DataFrame = [tweet_id: long, text: string ... 25 more fields]

# Data Cleaning - Schema
```

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.

Microsoft Azure databricks

silver_nb

```

File Edit View Run Help Python Tabs: ON Last edit was 2 hours ago
df: pyspark.sql.connect.DataFrame = [tweet_id: long, text: string ... 25 more fields]

# Data Cleaning - Schema
df = df \
    .withColumn("tweet_id", col("tweet_id").cast("long")) \
    .withColumn("author_id", col("author_id").cast("long")) \
    .withColumn("author_followers", col("author_followers").cast("long")) \
    .withColumn("author_following", col("author_following").cast("long")) \
    .withColumn("author_tweet_count", col("author_tweet_count").cast("long")) \
    .withColumn("impression_count", col("impression_count").cast("long")) \
    .withColumn("retweet_count", col("retweet_count").cast("int")) \
    .withColumn("like_count", col("like_count").cast("int")) \
    .withColumn("reply_count", col("reply_count").cast("int")) \
    .withColumn("quote_count", col("quote_count").cast("int")) \
    .withColumn("month", col("month").cast("string")) \
    .withColumn("year", col("year").cast("int")) \
    .withColumn("created_at", col("created_at").cast("timestamp")) \
    .withColumn("collected_at", col("collected_at").cast("timestamp"))

df: pyspark.sql.connect.DataFrame = [tweet_id: long, text: string ... 25 more fields]

# Verifying Schema
df.printSchema()
-- author_following: long (nullable = true)
-- author_tweet_count: long (nullable = true)

```

Microsoft Azure databricks

silver_nb

```

File Edit View Run Help Python Tabs: ON Last edit was 2 hours ago
Python

Enrichment

df = df \
    .withColumn("tweet_date", to_date(col("created_at"))) \
    .withColumn("tweet_hour", hour(col("created_at"))) \
    .withColumn("tweet_week", weekofyear(col("created_at"))) \
    .withColumn("tweet_day_of_week", dayofweek(col("created_at"))) \
    .withColumn("tweet_month_num", month(col("created_at")))

df: pyspark.sql.connect.DataFrame = [tweet_id: long, text: string ... 30 more fields]

df = df.withColumn('tweet_year', col('year')) \
    .drop('month', 'year')

df: pyspark.sql.connect.DataFrame = [tweet_id: long, text: string ... 29 more fields]

df = df.withColumn('state', lit('Tamil Nadu'))

df: pyspark.sql.connect.DataFrame = [tweet_id: long, text: string ... 30 more fields]

```

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

```

File Edit View Run Help Python Tabs: ON Last edit was 2 hours ago
+ Run all Connect Schedule (1) Share

df = pyspark.sql.connect.DataFrame = [tweet_id: long, text: string ... 30 more fields]

df = df.withColumn("sentiment_score",
    when(col("sentiment") == "positive", 1)
    .when(col("sentiment") == "negative", -1)
    .otherwise(0)
)

df = pyspark.sql.connect.DataFrame = [tweet_id: long, text: string ... 31 more fields]

df_mapping = spark.read.json("abfss://bronze@apoliticalparties.dfs.core.windows.net/party_mapping.json")

df_mapping = pyspark.sql.connect.DataFrame = [_corrupt_record: string, party: string ... 2 more fields]

df_mapping = df_mapping.drop('_corrupt_record')

df_mapping = pyspark.sql.connect.DataFrame = [party: string, party_category: string ... 1 more field]

df = df.join(df_mapping, on="party", how="left")

df = pyspark.sql.connect.DataFrame = [party: string, tweet_id: long ... 33 more fields]

```

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

```

File Edit View Run Help Python Tabs: ON Last edit was 2 hours ago
+ Run all Connect Schedule (1) Share

df = pyspark.sql.connect.DataFrame = [party: string, tweet_id: long ... 33 more fields]

df = df.withColumn("engagement",
    col("retweet_count") + col("like_count") + col("reply_count") + col("quote_count")) \
    .withColumn("weighted_sentiment",
    col("sentiment_score") * col("impression_count"))

df = pyspark.sql.connect.DataFrame = [party: string, tweet_id: long ... 35 more fields]

Writing Data

df.writeStream.format("delta") \
    .outputMode("append") \
    .option("checkpointlocation", "abfss://silver@apoliticalparties.dfs.core.windows.net/checkpoint") \
    .trigger(once = True) \
    .option("path", "abfss://silver@apoliticalparties.dfs.core.windows.net/politicalparties") \
    .toTable("ata_political.silver.political")

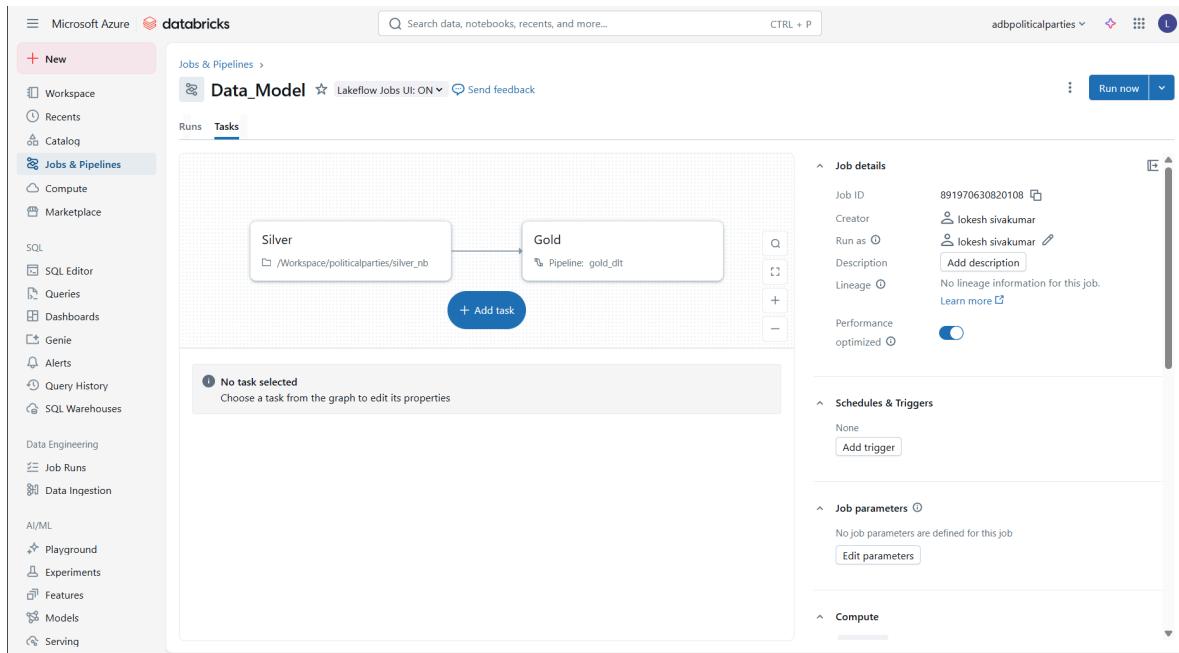
See performance (1)
Stream stopped...


```

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 Pipeline Editor for the 'gold_dlt' pipeline. The left sidebar lists various assets like 'silver_cleaned.py', 'dim_date.py', etc. The main area displays the pipeline configuration and a code editor containing Python code for data cleaning:

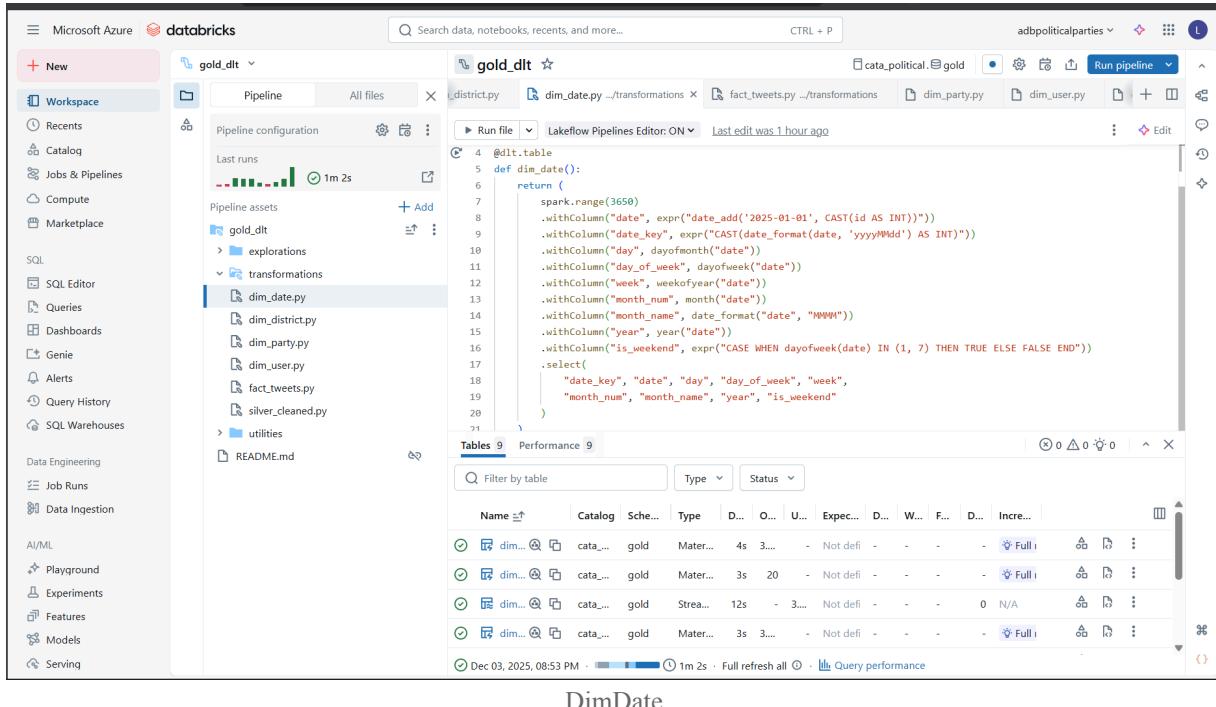
```
1 import dlt
2 from pyspark.sql.functions import col
3
4 @dlt.table()
5 def silver_cleaned():
6     return (
7         spark.read.format("delta")
8         .load("abfss://silver@sapopoliticalparties.dfs.core.windows.net/politicalparties")
9         .drop("_rescued_data")
10    )
11
```

Below the code editor, a table titled 'Tables' shows performance metrics for various tables, including 'dim_date', 'dim_party', 'dim_user', and 'fact_tweets'. The table includes columns for Name, Catalog, Schema, Type, and various performance metrics like Duration, Throughput, and Error rate.

SilverCleaned

Data Modeling

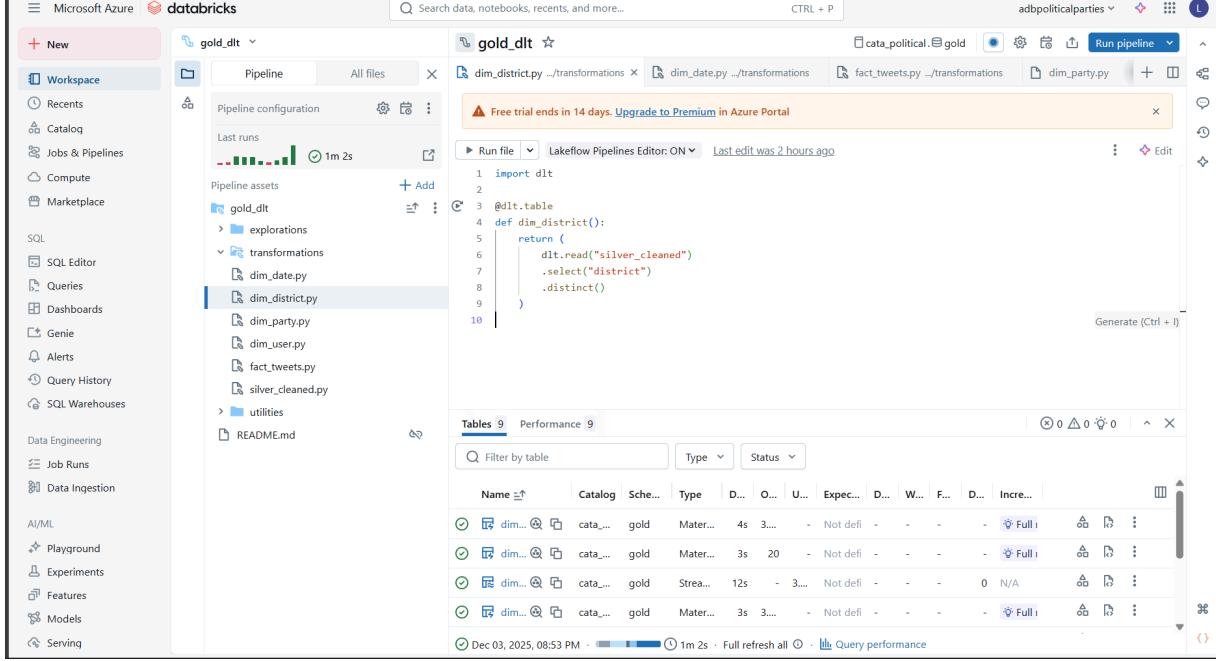
Here, I have transformed the data into a star schema format.



The screenshot shows the Databricks Pipeline Editor interface for a pipeline named "gold_dlt". The left sidebar lists various workspace sections like Recents, Catalog, and Compute. The main area displays the "Pipeline configuration" tab, which includes a "Last runs" section showing a single run took 1m 2s, and a "Pipeline assets" section listing several Python files under the "transformations" folder: dim_date.py, dim_district.py, dim_party.py, dim_user.py, fact_tweets.py, and silver_cleaned.py. Below this is a "Tables" section showing a list of tables with their details. A status bar at the bottom indicates a full refresh was completed at 08:53 PM on Dec 03, 2025.

```
4 @dlt.table
5 def dim_date():
6     return (
7         spark.range(3650)
8             .withColumn("date", expr("date_add('2025-01-01', CAST(id AS INT))"))
9             .withColumn("date_key", expr("CAST(date_format(date, 'yyyyMMdd') AS INT)"))
10            .withColumn("day", dayofmonth("date"))
11            .withColumn("day_of_week", dayofweek("date"))
12            .withColumn("week", weekofyear("date"))
13            .withColumn("month_num", month("date"))
14            .withColumn("month_name", date_format("date", "MMMM"))
15            .withColumn("year", year("date"))
16            .withColumn("is_weekend", expr("CASE WHEN dayofweek(date) IN (1, 7) THEN TRUE ELSE FALSE END"))
17        .select(
18            "date_key", "date", "day", "day_of_week", "week",
19            "month_num", "month_name", "year", "is_weekend"
20        )
21    )
```

DimDate



This screenshot shows the Databricks Pipeline Editor for the "gold_dlt" pipeline, specifically focusing on the "dim_district.py" transformation. It features a similar layout to the previous screenshot, with the left sidebar and pipeline configuration tab. The "Tables" section at the bottom shows the same list of tables as the first screenshot. A message in the center of the editor states "Free trial ends in 14 days. Upgrade to Premium in Azure Portal". The code for the dim_district transformation is visible in the editor:

```
1 import dlt
2
3 @dlt.table
4 def dim_district():
5     return (
6         dlt.read("silver_cleaned")
7             .select("district")
8             .distinct()
9     )
10
```

DimDistrict

Microsoft Azure databricks

gold_dlt

Pipeline configuration

Last runs

Pipeline assets

gold_dlt

transformations

dim_party.py

```

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    )

```

Tables 9 Performance 9

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

DimParty

Microsoft Azure databricks

gold_dlt

Pipeline configuration

Last runs

Pipeline assets

gold_dlt

transformations

dim_party.py

```

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

```

Tables 9 Performance 9

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

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.

Microsoft Azure gold_dlt

Search data, notebooks, recents, and more... CTRL + P

adbpoliticalparties

New

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
- Serving

Pipeline Pipeline configuration All files gold_dlt

Last runs 1m 2s

Pipeline assets gold_dlt explorations transformations

```

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             )

```

Tables 9 Performance 9

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

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

DimUser

Microsoft Azure gold_dlt

Search data, notebooks, recents, and more... CTRL + P

adbpoliticalparties

New

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
- Serving

Pipeline Pipeline configuration All files gold_dlt

Last runs 1m 2s

Pipeline assets gold_dlt explorations transformations

```

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,

```

Tables 9 Performance 9

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

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

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 Pipeline Editor interface. The left sidebar contains navigation links for Microsoft Azure, Databricks, Workspace, Recents, Catalog, Jobs & Pipelines, Compute, Marketplace, SQL, SQL Editor, Queries, Dashboards, Genie, Alerts, Query History, and SQL Warehouses. The Data Engineering section includes Job Runs, Data Ingestion, AI/ML, Playground, Experiments, Features, Models, and Serving.

The main area displays the "gold_dlt" pipeline. The pipeline configuration shows "Last runs" (1m 2s ago) and "Pipeline assets" (fact_tweets.py selected). The code editor shows the Python script for "fact_tweets.py":

```
4 def fact_tweets_stg():
5     df = dlt.read("silver_cleaned")
6     return df.select(
7         "tweet_id",
8         "tweet_date",
9         "tweet_hour",
10        "tweet_week",
11        "tweet_day_of_week",
12        "tweet_month_num",
13        "tweet_year",
14        "district",
15        "party",
16        "sentiment",
17        "sentiment_score",
18        "engagement",
19        "weighted_sentiment",
20        "impression_count",
21        "+button_id"
```

The pipeline status bar at the bottom indicates "Dec 03, 2025, 08:53 PM · 1m 2s · Full refresh all · Query performance".

The screenshot shows the Microsoft Azure Databricks workspace interface. The left sidebar contains navigation links for Microsoft Azure, Databricks, 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 a pipeline named "gold_dlt". The pipeline configuration shows "Last runs" with a duration of "1m 2s". Pipeline assets include "gold_dlt" (with "explorations", "transformations" like "dim_date.py", "dim_district.py", "dim_party.py", "dim_user.py", and "fact_tweets.py"), "silver_cleaned.py", and "utilities". A README.md file is also present. The "fact_tweets.py" file is currently selected, showing Python code for creating streaming tables:

```
20     "impression_count",
21     "author_id",
22     "collected_at"
23 )
24
25 dlt.create_streaming_table("fact_tweets")
26
27 dlt.create_auto_cdc_flow(
28     target="fact_tweets",
29     source="fact_tweets_stg",
30     keys=[("tweet_id")],
31     sequence_by="collected_at",
32     stored_as_scd_type=1,
33     track_history_except_column_list=None,
34     name=None,
35     once=False
36 )
```

The "Tables" and "Performance" tabs are visible at the bottom, along with a "Filter by table" search bar and status indicators.

Delta Live Tables

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

The screenshot shows the Databricks Jobs & Pipelines interface for the 'Data_Model' pipeline. The pipeline structure is as follows:

- Silver**: The first layer.
- Gold**: The second layer.
- Model**: The third layer.

A successful run was completed on **Dec 03**, 2025, with a total duration of **1m 58s**. The run details are:

Start time	Run ID	Launched	Duration	Status	Error code	Run param...
Dec 03, 2025, 0...	112296412...	Manually	1m 58s	Succ...		

Job details for the job ID 891970630820108:

- Creator: lokesh sivakumar
- Run as: lokesh sivakumar
- Description: No lineage information for this job.
- Performance optimization: Enabled

Schedules & Triggers: None

Job parameters: No job parameters are defined for this job.

Compute: Not specified.

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.