Azure Databricks end to end Project with Unity Catalog CI/CD

© Shubham Wadekar

Real-Time Data Lakehouse for Traffic and Roads Analytics

Project Overview

This project builds a real-time data pipeline using Azure Data Lake and Delta Lake to process and analyze traffic and roads datasets. Data is manually loaded into a landing zone to simulate incremental ingestion. It then flows into the bronze layer using Spark Structured Streaming with Auto Loader, capturing all raw records.

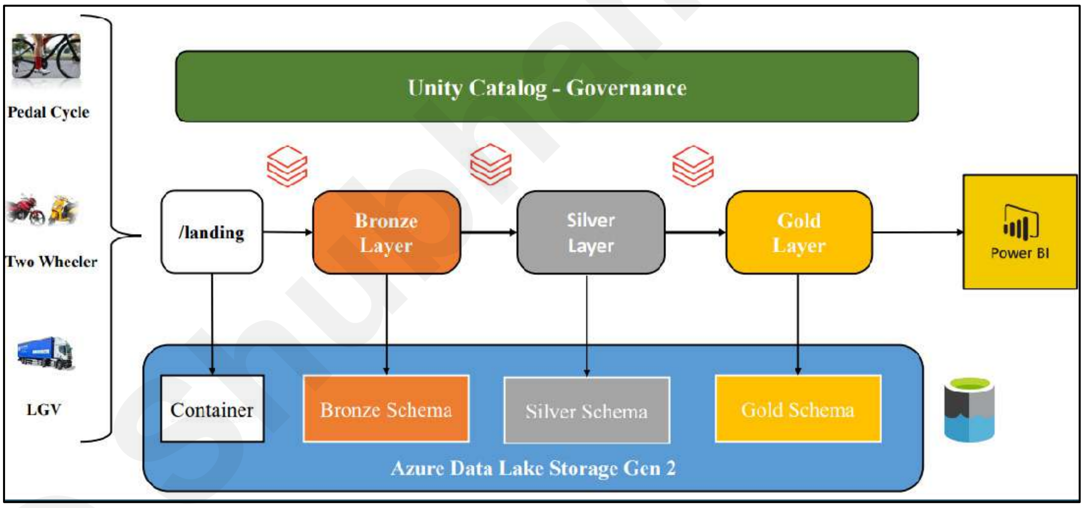
From the bronze layer, only new data is transformed and moved to the silver layer for cleaning and enrichment. The final curated data is stored in the gold layer as optimized tables or views, ready for reporting and data science.

Key technologies include:

- Delta Lake for reliable storage and ACID transactions
- Auto Loader for incremental data ingestion
- Power BI for reporting and visualization
- Azure DevOps for CI/CD
- Access controls to simulate enterprise-grade security

This end-to-end pipeline supports real-time analytics in a scalable, cloud-native environment.

Project Architecture

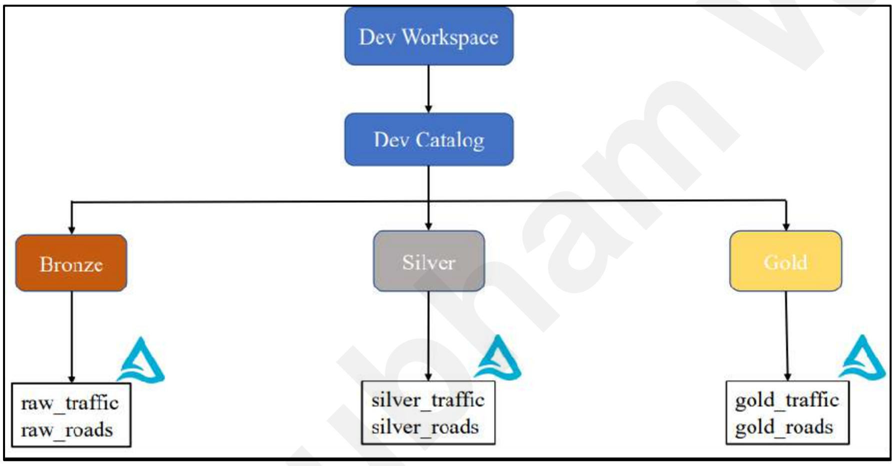


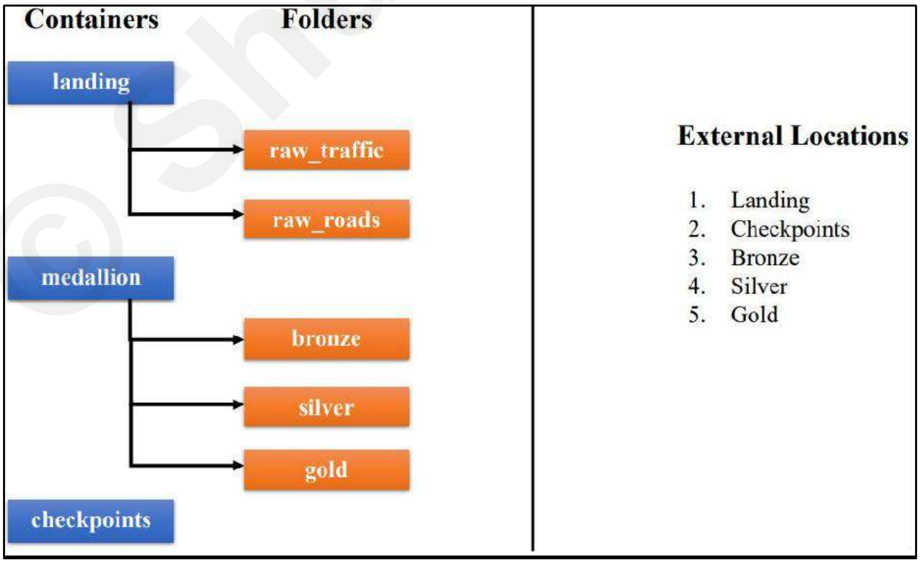
© Shubham Wadekar

The project uses a Medallion Architecture (Bronze → Silver → Gold) on Azure Databricks with Delta Lake and Unity Catalog for governance. Data flows through the following stages:

- Landing Zone: Raw traffic and roads data are manually ingested into Azure Data Lake Storage Gen2 (/landing folder) to simulate streaming input.
- Bronze Layer: Ingested incrementally using Auto Loader with Structured Streaming. Raw data is stored as Delta tables (raw_traffic, raw_roads) under the bronze schema.
- Silver Layer: Transforms include renaming columns, deriving Electric_Vehicles_Count, Motor_Vehicles_Count, and categorizing road types. Cleaned datasets (silver_traffic, silver_roads) are stored in the silver schema.
- Gold Layer: Final business logic is applied (e.g., Vehicle_Intensity), and tables are optimized for reporting (gold_traffic, gold_roads). These are consumed in Power BI for insights.
- Governance: Managed via Unity Catalog with a three-tier namespace (catalog.schema.table) and fine-grained access control.
- CI/CD: Implemented using Azure DevOps for deploying code and configurations across environments (Dev → UAT → Prod).

Project Setup

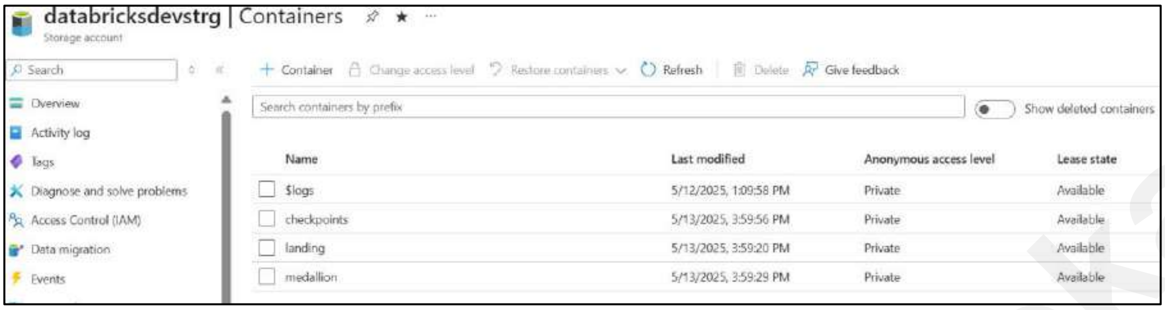




© Shubham Wadekar

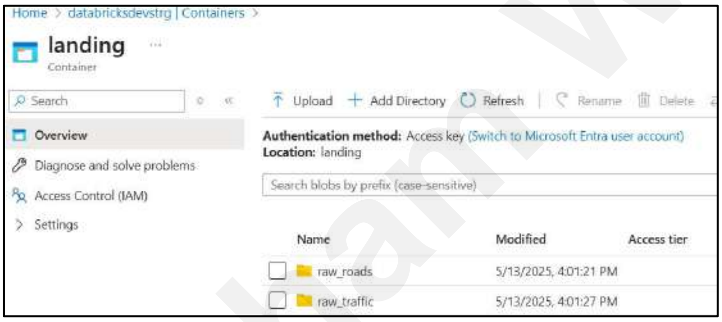
This project is organized using Azure Data Lake Storage Gen2, structured into containers and folders representing different stages of the data pipeline.

Containers and Folders

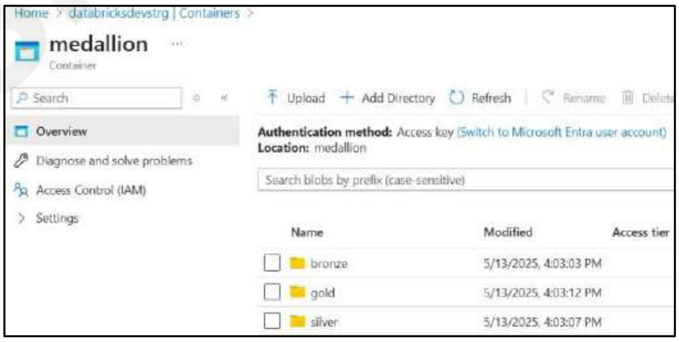


We have 3 containers in Azure Data Lake Storage:

1. Landing:
    a. Holds the raw input data.
    b. Contains 2 sub folders: raw_traffic (for traffic dataset), raw_roads (for road dataset).



2. Medallion:
    a. Implements the medallion architecture with subfolders.
    b. Bronze – stores raw ingested data
    c. Silver – stores cleaned and transformed data
    d. Gold – stores curated, business-ready data



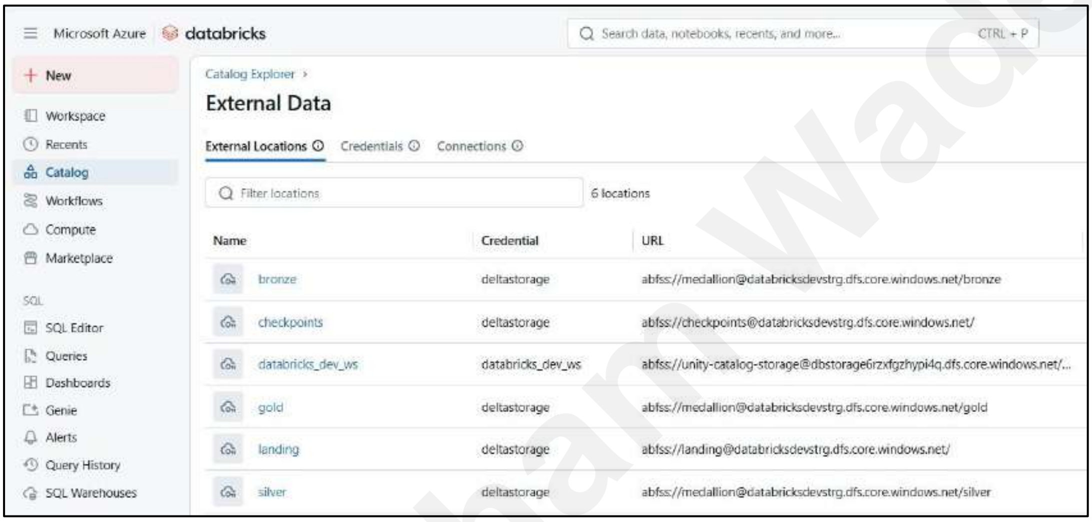
© Shubham Wadekar

3. Checkpoints:
    a. Stores streaming query checkpoints to support fault tolerance and exactly once processing in Structured Streaming.

External Locations

The following external locations are registered in Unity Catalog to enable secure and governed access to data:

1. Landing
2. Checkpoints
3. Bronze
4. Silver
5. Gold



Each location corresponds to a specific path in the data lake and is linked with appropriate storage credentials and access controls using Unity Catalog.

Data Sources

The datasets are from **Kaggle**. **Raw**

Traffic Dataset

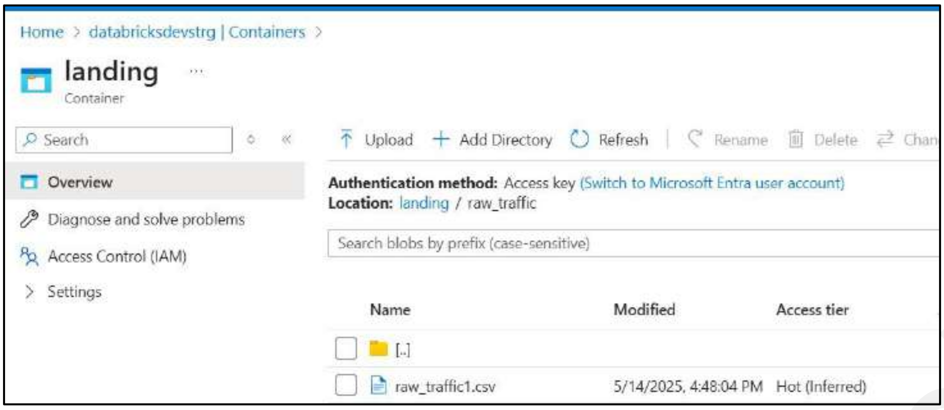
The Raw Traffic Dataset is one of the core inputs for this project. It has actual data collected by trained enumerators to feed data into road traffic estimates. It contains structured information collected from traffic monitoring points across UK roads. This dataset has a raw count for the number of vehicles of each type that flowed past at each point of day, broken by direction and an hour.

It has pedal cycles, 2-wheeler vehicles, buses and coaches, LGV (Large Good Vehicles) and HGV (Heavy goods vehicles), and electric vehicles. So, we need to find out at a given time within an hour, how many vehicles are recorded in the raw traffic count dataset. We will analyze the type of vehicle travelling at a given point along with roads.

© Shubham Wadekar

Source and Storage

- The dataset is manually placed in the /landing/raw_traffic folder of Azure Data Lake



Storage Gen2.

- It simulates real-time ingestion and is later processed incrementally using Spark Structured Streaming with Auto Loader.
- Once ingested, it is first stored in the bronze layer as the table raw_traffic.

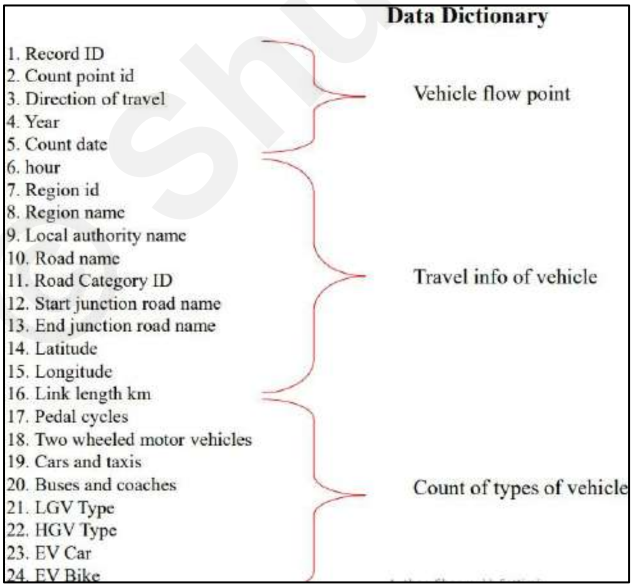
Purpose in the pipeline

This dataset serves multiple purposes:

- Provides the raw measurements of traffic flow, needed to calculate derived metrics.
- Enables tracking of hourly, daily, and yearly trends in vehicle movement.
- Acts as a base for data enrichment and transformation in the silver layer, eventually powering analytical dashboards and reports on the gold layer.

Data Dictionary

Data Dictionary has all column names. It defines what information that each column has.



Shubham Wadekar

Data Dictionary

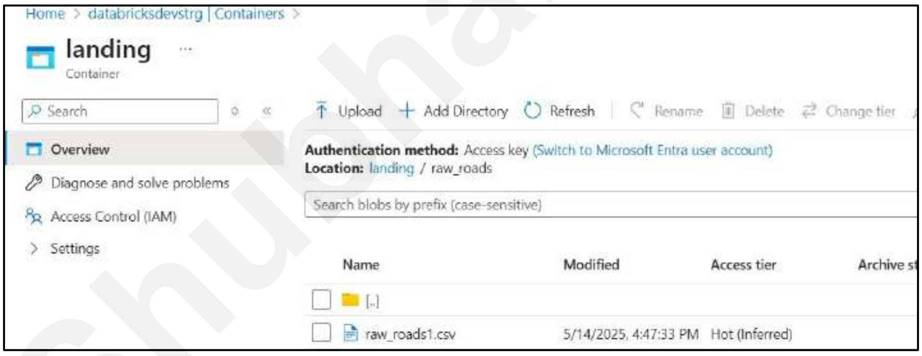
1. Record ID	=	Uniquely identifies a record
2. Count point id	=	A unique reference for the road link
3. Direction of travel	=	Direction of travel
4. Year	=	Year it happened
5. Count date	=	The date when the actual count took place
6. hour	=	Hour 7 represents from 7am to 8am, and 17 tells from 5pm to 6pm
7. Region id	=	Website region identifier
8. Region name	=	The name of the Region that travel took place
9. Local authority name	=	Local authority that region
10. Road name	=	This is the road name (for instance M25 or A3).
11. Road Category ID	=	Uniquely identifies road ID
12. Start junction road name	=	The road name of the start junction of the link
13. End junction road name	=	The road name of the end junction of the link
14. Latitude	=	Latitude of the Location
15. Longitude	=	Longitude of the Location
16. Link length km	=	Total length of the network road link
17. Pedal cycles	=	Counts for pedal cycles
18. Two wheeled motor vehicles	=	Counts of Two wheeled motor vehicles
19. Cars and taxis	=	Counts of Cars and taxis
20. Buses and coaches	=	Counts of Buses and coaches
21. LGV Type	=	Counts of LGV Type
22. HGV Type	=	Counts of HGV Type
23. EV Car	=	Counts of EV Car
24. EV Bike	=	Counts of EV Bike

Raw Roads Dataset

The raw roads dataset defines the road category. It provides essential metadata about the road network across different regions. It includes classifications, measurements, and summaries of road segments, which are later used for enriching traffic data and performing spatial analysis.

<u>Source and storage</u>

- The dataset is manually ingested into the /landing/raw_roads folder within Azure Data Lake Storage Gen2.



- It simulates external data ingestion and is incrementally loaded into the bronze layer using Spark Structured Streaming with Auto Loader.
- Stored as a Delta table named raw_roads in the bronze schema.

<u>Purpose in the pipeline</u>

The Raw Roads Dataset:

- Provides road classification and geographical context for traffic analysis.
- Helps join with traffic datasets using common region or road category IDs.

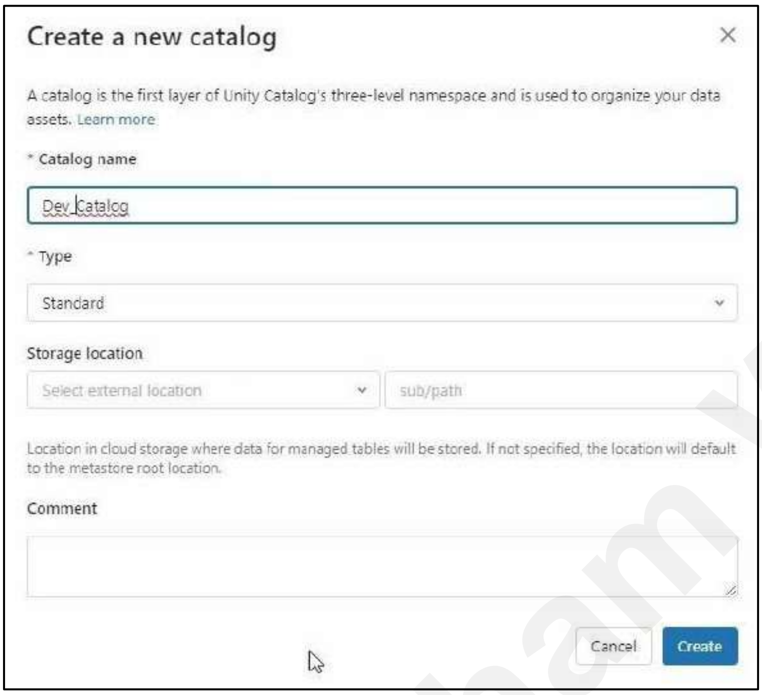
© Shubham Wadekar

- Supports the derivation of road type attributes used in visualization and aggregation in the gold layer.

The common link or common column from both the datasets is Road Category.

Creating Databricks Dev Catalog

We created the dev workspace for Azure Databricks (**databricks-dev-ws**).



Access Connector for Databricks

In this project, the Access Connector for Azure Databricks is used to securely access Azure Data Lake Storage Gen2, where all data layers (landing, bronze, silver, gold, checkpoints) are stored. It enables Databricks to read **raw traffic** and **raw roads** datasets and write Delta tables without using storage keys, by leveraging managed identity authentication. This ensures secure, role-based access control and is required for integrating with Unity Catalog's external locations.

© Shubham Wadekar

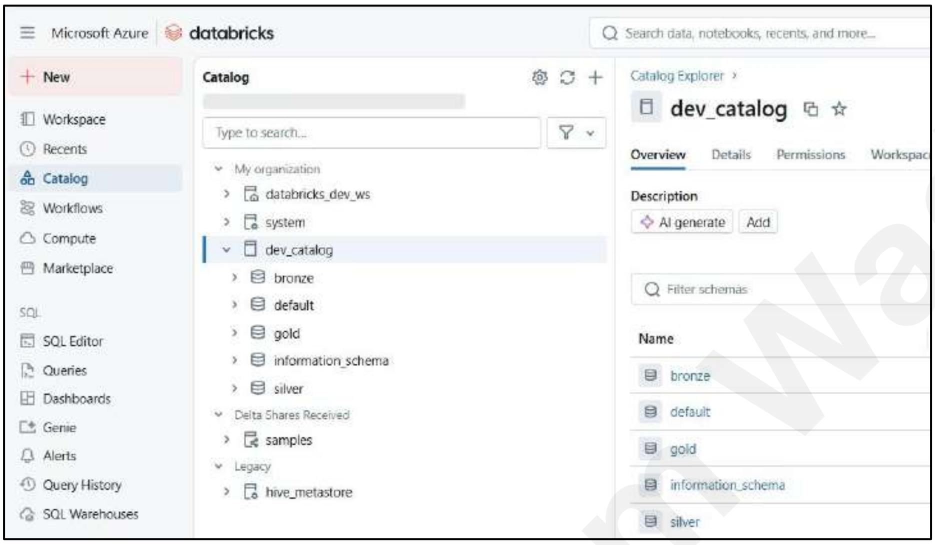
Metastore Creation

A metastore is a top-level container for data in Unity Catalog. Within a metastore, Unity Catalog provides a 3-level namespace for organizing data (catalogs, schemas, tables/views).

If we do not assign the workspace to a metastore we will not be able to create a catalog or schema. After assigning the workspace to the metastore we need to enable the Unity Catalog.

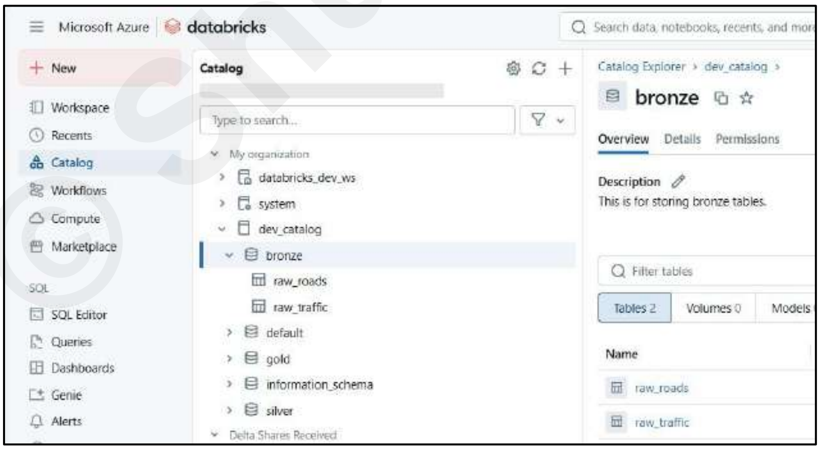
Creating all the Schemas in Dev Catalog

We have created 3 schemas in the dev catalog (bronze, silver, gold).



Ingestion to Bronze Layer

The bronze layer is the first structured zone in the medallion architecture where raw data is ingested from the landing zone and stored as Delta tables. It serves as the source of the truth, capturing unprocessed data exactly as received.



© Shubham Wadekar

Ingestion Process

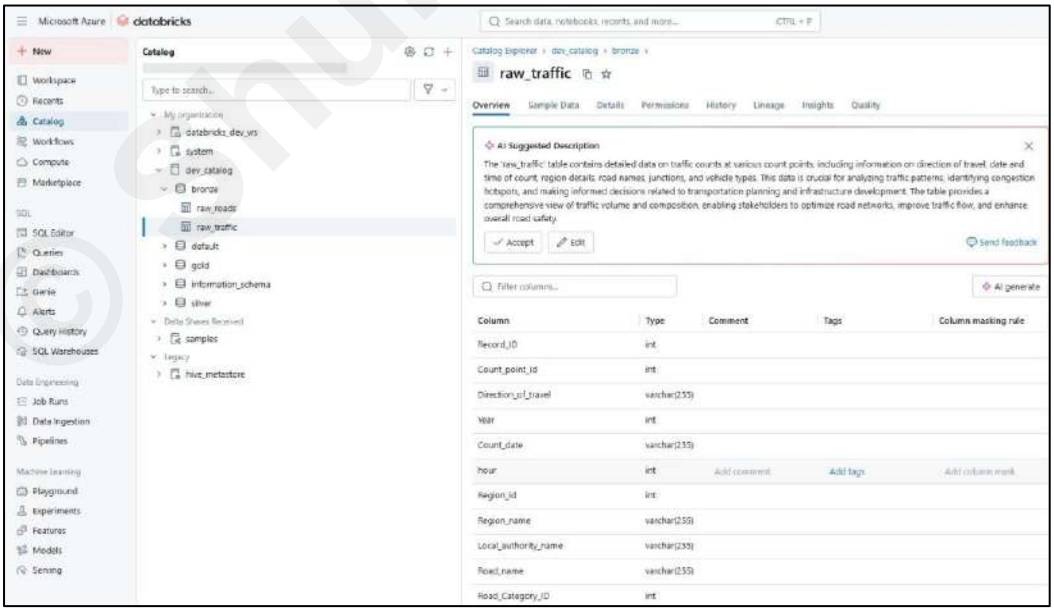
- Source:
    - Data is manually placed in:
        - /landing/raw_traffic (for traffic data)
        - /landing/raw_roads (for road data)

- Streaming Ingestion with Auto Loader:
    - Spark reads incoming files using Auto Loader. It improves pipeline efficiency by only processing newly arrived data.
    - Used to incrementally ingest raw traffic and road CSV files from the /landing folder.
    - We have created 2 autoloaders. One for raw_roads and the other for raw_traffic.
    - Enables real-time data ingestion into the bronze layer using Structured Streaming.
    - Reads data using .format("cloudFiles") with cloudFiles.format = "csv".
    - Stores schema information using cloudFiles.schemaLocation for schema inference.
    - Tracks progress using a checkpoint directory to ensure fault tolerance and supports automatic detection of new files, eliminating the need for manual triggers.
    - Loads data into Delta tables: bronze.raw_traffic and bronze.raw_roads.

- Schema: Bronze

- Tables created:
    - raw_traffic
    - raw_roads

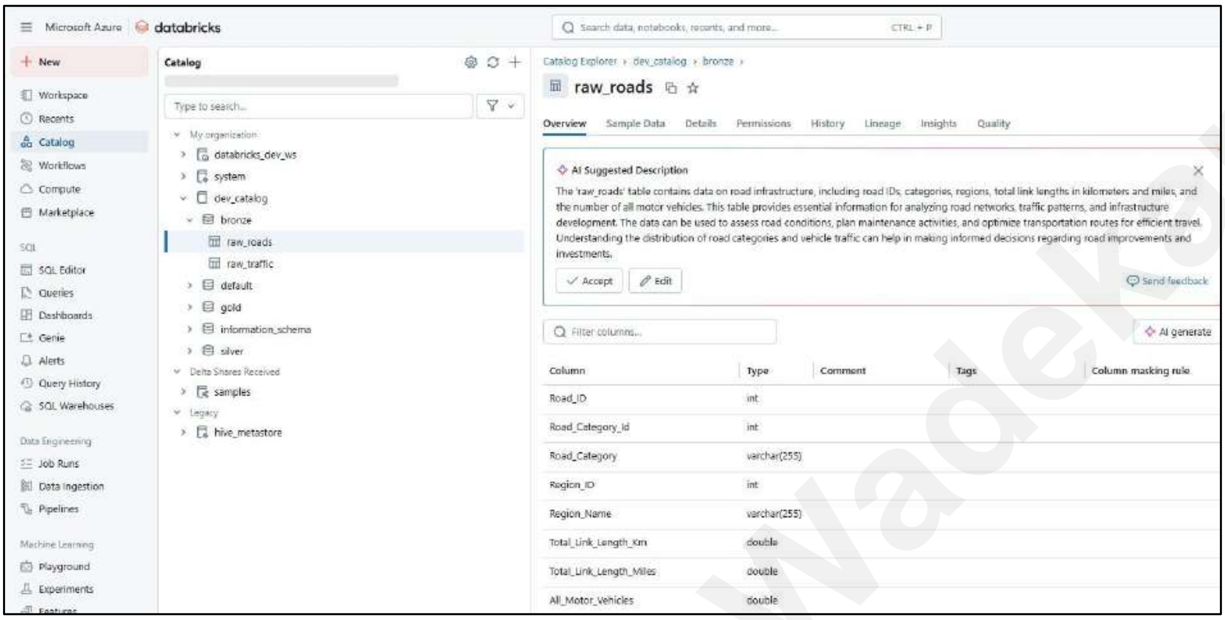
Ingestion of raw_traffic table



© Shubham Wadekar

We have ingested the raw_traffic table into the bronze schema within the dev_catalog in the Databricks workspace (dataricks_dev_ws).

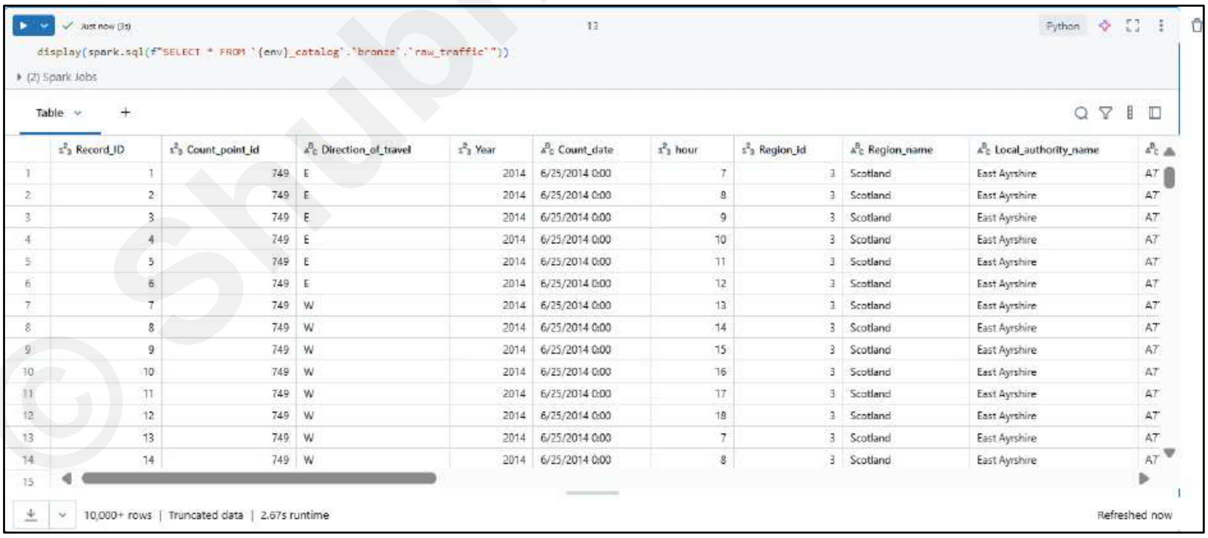
Ingestion of raw_roads table



We have ingested the raw_roads table into the bronze schema within the dev_catalog in the Databricks workspace (dataricks_dev_ws).

Loading data to Bronze Tables

Data written for raw_traffic:



After defining the schema and reading the raw_traffic CSV file from the landing zone in using **Auto Loader**, the data was successfully written to the bronze layer in Delta format.

Data written for raw_roads:

```
display(spark.sql(f"SELECT * FROM `{env}_catalog`.`bronze`.`raw_roads`"))
```

▸ (2) Spark Jobs

Table

	Road_ID	Road_Category_Id	Road_Category	Region_ID	Region_Name	Total_Link_Length_Km	Total_Link_Length_Miles	All_Motor_Vehicles
1	1	1	TM	1	South West	301.339	187.24	346584
2	2	3	TA	1	South West	993.586	617.39	348471
3	3	4	PA	1	South West	3874.924	2407.77	779400
4	4	5	M	1	South West	43581.7	27080.41	911200
5	5	1	TM	2	East Midlands	178.609	110.98	273666
6	6	3	TA	2	East Midlands	1219.231	757.6	493670
7	7	4	PA	2	East Midlands	2571.21	1597.68	553572
8	8	5	M	2	East Midlands	26712.7	16598.5	708336
9	9	1	TM	3	Scotland	335.263	208.32	248546
10	10	3	TA	3	Scotland	2820.302	1752.45	505292
11	11	4	PA	3	Scotland	7405.483	4601.55	651443
12	12	5	M	3	Scotland	47971.6	29808.17	765643
13	13	1	TM	4	Wales	130.291	80.96	130658
14	14	3	TA	4	Wales	1509.635	938.04	312394
15								

76 rows | 2.33s runtime

Refreshed now

After defining the schema and reading the raw_roads CSV file from the landing zone using **Auto Loader**, the data was written to the bronze layer in Delta format.

Proving Autoloader handles incremental loading

Autoloader tracks based on the checkpoint location, and it is going to write this data incrementally by reading the only newly added rows.

```
%sql

SELECT COUNT(*) FROM `dev_catalog`.`bronze`.`raw_traffic`
```

▸ (3) Spark Jobs

▸ _sqldf: pyspark.sql.dataframe.DataFrame = [count(1): long]

Table

	count(1)
1	18546

- Checking the row count of the raw_traffic data

- The same we can say that the last record_id column value would also be the same

```
%sql

SELECT MAX(Record_ID) FROM `dev_catalog`.`bronze`.`raw_traffic`
```

▸ (3) Spark Jobs

▸ _sqldf: pyspark.sql.dataframe.DataFrame = [max(Record_ID): integer]

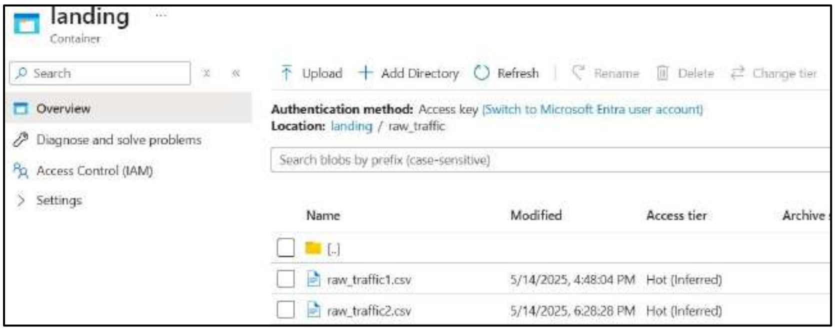
Table

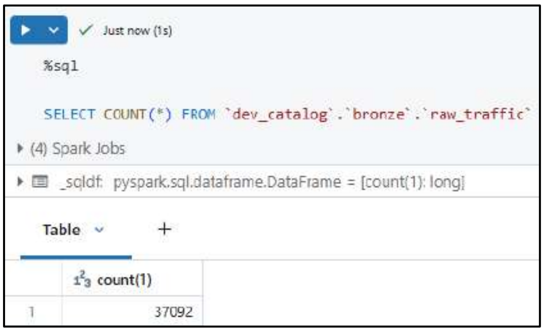
	max(Record_ID)
1	18546

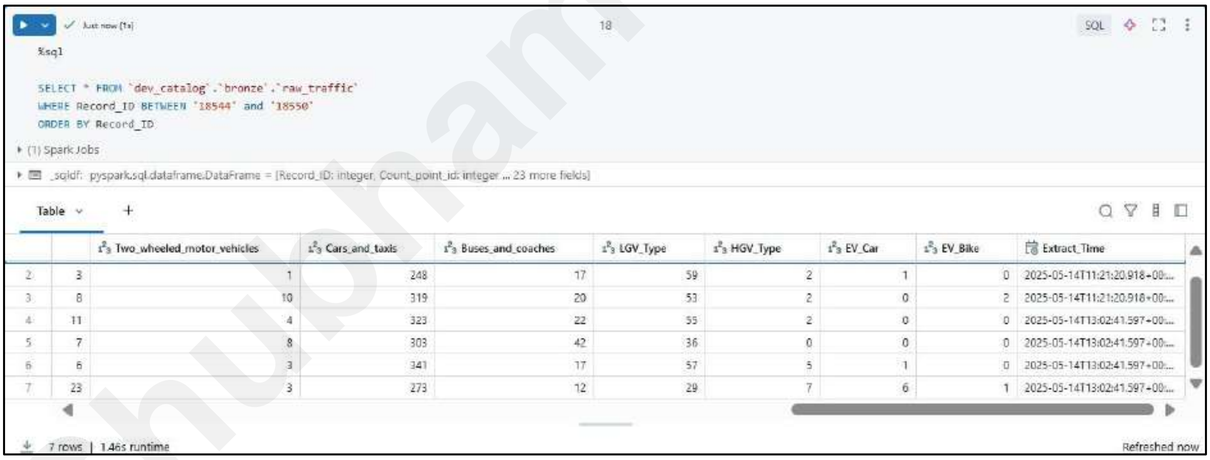
number as the above row count.

© Shubham Wadekar

- Now if we upload another file, those many new records will now be having new extract time value after 18546 records. There will be difference in the extract time value.







The count changed from 18546 to 37092. We can check the timestamp between the last few record_id till some of the newly added record_id.

- So, this proves that this is going to take that data based on a micro batch. For each micro batch, it is going to process all records which are available and is going to write the last record information somewhere in the checkpoint. When we upload another data, it goes to the checkpoint, and it is going to see where exactly the previous load was done and is going to compare that and then do the next load.
This proves that autoloader is capable to do the incremental loading.

- To run the notebook to process the newly added data, we need to monitor if there is any new file available. We can have it in a scheduled manner as well like twice a day or thrice a day. For now, we have just manually run the notebook to check the results.

© Shubham Wadekar

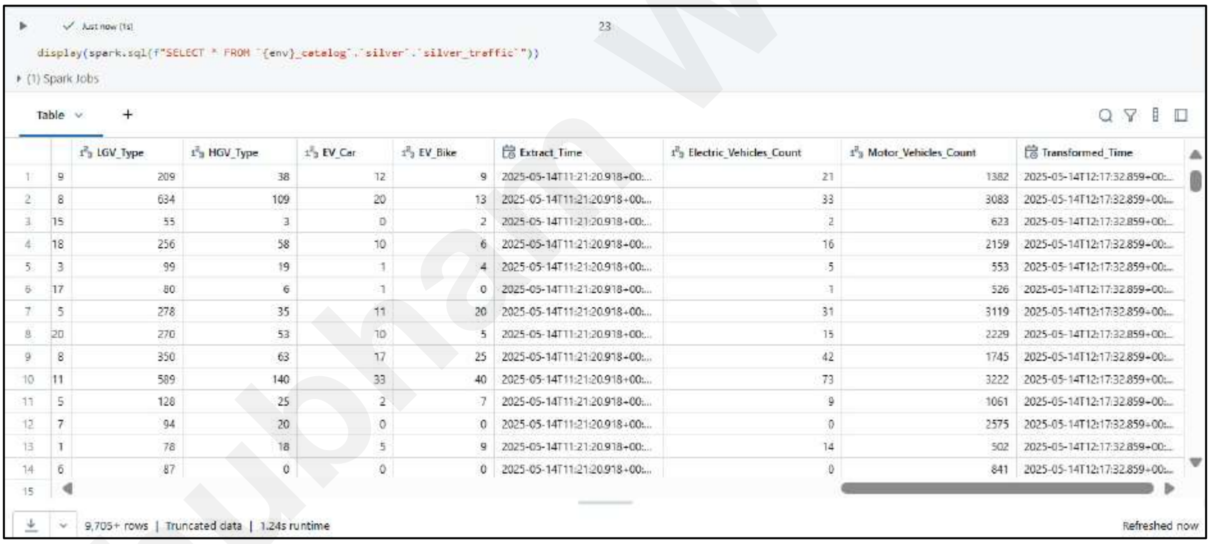
Transforming Data into Silver Layer

Raw Traffic and Raw Roads data from the bronze layer is cleaned and enriched here.

- Schema: Silver
- Tables: silver_traffic, silver_roads

Transformations on raw_traffic   silver_traffic:

- Renamed columns for easier querying and readability (e.g., Count point id → Count_point_id).
- Removed duplicates.
- Created Electric_Vehicles_Count = EV_Car + EV_Bike. It combines electric vehicle types to get total EV presence at a location.
- Created Motor_Vehicles_Count = Two_wheeled_motor_vehicles + Cars_and_taxis + Buses_and_coaches + LGV_Type + HGV_Type + Electric_Vehicles_Count.
  It calculates the total number of motorized vehicles for a given record.
- Derived Vehicle_Intensity = Motor_Vehicles_Count / Link_length_km to measure traffic density.
- Added timestamp columns like Extract_Time (from bronze) to track ingestion time.



Just now (1s)   23

```
display(spark.sql(f"SELECT * FROM `{env}_catalog`.`silver`.`silver_traffic`"))
```

(1) Spark Jobs

Table

		LGV_Type	HGV_Type	EV_Car	EV_Bike	Extract_Time	Electric_Vehicles_Count	Motor_Vehicles_Count	Transformed_Time
1	9	209	38	12	9	2025-05-14T11:21:20.918+00:...	21	1382	2025-05-14T12:17:32.859+00:...
2	8	634	109	20	13	2025-05-14T11:21:20.918+00:...	33	3083	2025-05-14T12:17:32.859+00:...
3	15	55	3	0	2	2025-05-14T11:21:20.918+00:...	2	623	2025-05-14T12:17:32.859+00:...
4	18	256	58	10	6	2025-05-14T11:21:20.918+00:...	16	2159	2025-05-14T12:17:32.859+00:...
5	3	99	19	1	4	2025-05-14T11:21:20.918+00:...	5	553	2025-05-14T12:17:32.859+00:...
6	17	80	6	1	0	2025-05-14T11:21:20.918+00:...	1	526	2025-05-14T12:17:32.859+00:...
7	5	278	35	11	20	2025-05-14T11:21:20.918+00:...	31	3119	2025-05-14T12:17:32.859+00:...
8	20	270	53	10	5	2025-05-14T11:21:20.918+00:...	15	2229	2025-05-14T12:17:32.859+00:...
9	8	350	63	17	25	2025-05-14T11:21:20.918+00:...	42	1745	2025-05-14T12:17:32.859+00:...
10	11	589	140	33	40	2025-05-14T11:21:20.918+00:...	73	3222	2025-05-14T12:17:32.859+00:...
11	5	128	25	2	7	2025-05-14T11:21:20.918+00:...	9	1061	2025-05-14T12:17:32.859+00:...
12	7	94	20	0	0	2025-05-14T11:21:20.918+00:...	0	2575	2025-05-14T12:17:32.859+00:...
13	1	78	18	5	9	2025-05-14T11:21:20.918+00:...	14	502	2025-05-14T12:17:32.859+00:...
14	6	87	0	0	0	2025-05-14T11:21:20.918+00:...	0	841	2025-05-14T12:17:32.859+00:...
15									

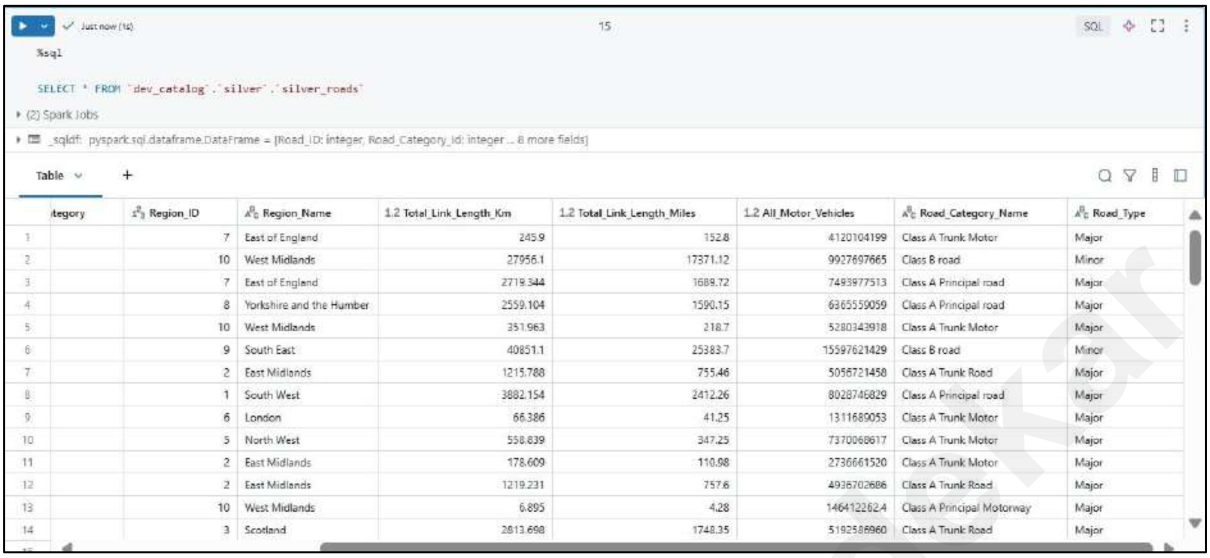
9,705+ rows | Truncated data | 1.24s runtime   Refreshed now

Transformations on raw_roads   silver_roads:

- Renamed fields (e.g., Road category → Road_category).
- Created Road_Category_Name using mappings. It converts road category codes into human-readable names to improve dashboard clarity.
  - TA → Class A Trunk Road
  - TM → Class A Trunk Motorway
  - PA → Class A Principal Road
  - PM → Class A Principal Motorway
  - M → Class B Road
- Derived Road_Type groups road categories into broader classifications for filtering and aggregation.
  - If Road_Category_Name contains "Class A" → "Major"

© Shubham Wadekar

- If Road_Category_Name contains "Class B" → "Minor"



```
%sql

SELECT * FROM `dev_catalog`.`silver`.`silver_roads`
```

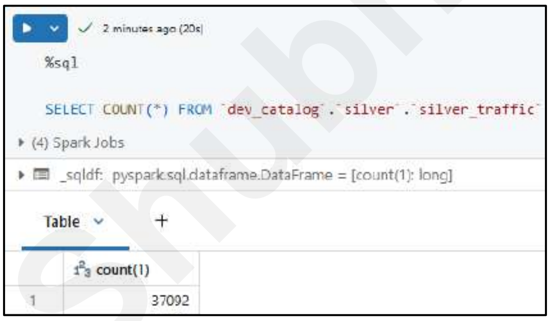
(2) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [Road_ID: integer, Road_Category_Id: integer ... 8 more fields]

	tegory	Region_ID	Region_Name	Total_Link_Length_Km	Total_Link_Length_Miles	All_Motor_Vehicles	Road_Category_Name	Road_Type
1		7	East of England	245.9	152.8	4120104199	Class A Trunk Motor	Major
2		10	West Midlands	27956.1	17371.12	9927697665	Class B road	Minor
3		7	East of England	2719.344	1689.72	7493977513	Class A Principal road	Major
4		8	Yorkshire and the Humber	2559.104	1590.15	6365559059	Class A Principal road	Major
5		10	West Midlands	351.963	218.7	5280343918	Class A Trunk Motor	Major
6		9	South East	40851.1	25383.7	15597621429	Class B road	Minor
7		2	East Midlands	1215.788	755.46	5056721458	Class A Trunk Road	Major
8		1	South West	3882.154	2412.26	8028746829	Class A Principal road	Major
9		6	London	66.386	41.25	1311689053	Class A Trunk Motor	Major
10		5	North West	558.839	347.25	7370068617	Class A Trunk Motor	Major
11		2	East Midlands	178.609	110.98	2736661520	Class A Trunk Motor	Major
12		2	East Midlands	1219.231	757.6	4936702686	Class A Trunk Road	Major
13		10	West Midlands	6.895	4.28	1464122624	Class A Principal Motorway	Major
14		3	Scotland	2813.698	1748.35	5192586960	Class A Trunk Road	Major

<u>Transformations on Incrementally Loaded Data</u>

Here we can see that only the newly added records were taken to process the data. It is because the bronze tables can have thousands of records every time, the incremental loading will be taken place from the landing zone to bronze, where incremental data will be appended to the bronze table and in point of time somewhere the records may be a million records. And if we are trying to do this silver layer transformation by creating a new column and applying the data that should not be applied on the entire data set each time, this should be applied only to the changed data, which means the rows which are newly added.



```
%sql

SELECT COUNT(*) FROM `dev_catalog`.`silver`.`silver_traffic`
```

(4) Spark Jobs

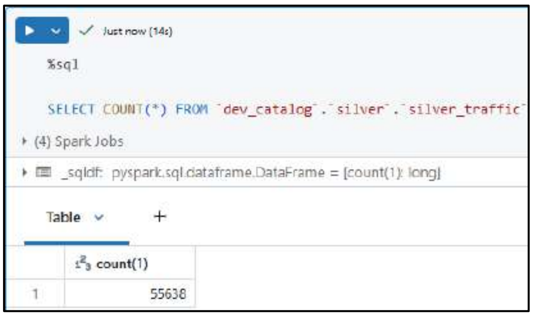
_sqldf: pyspark.sql.dataframe.DataFrame = [count(1): long]

	count(1)
1	37092

- Checking the count of the current records. It is 37092 after adding the 2nd traffic file.

- When we add the 3rd traffic file, the count changes to 55638. Now when we query the silver transformed data for traffic data, we can see the changes in transformed time for the new records only. The previous loaded data was the previous time when it was transformed. **So, this proves that the data was transformed incrementally and did not transform the old data.** This is possible because we are using the spark structure streaming and there is a delta lake for this table.

© Shubham Wadekar



Just now (14s)

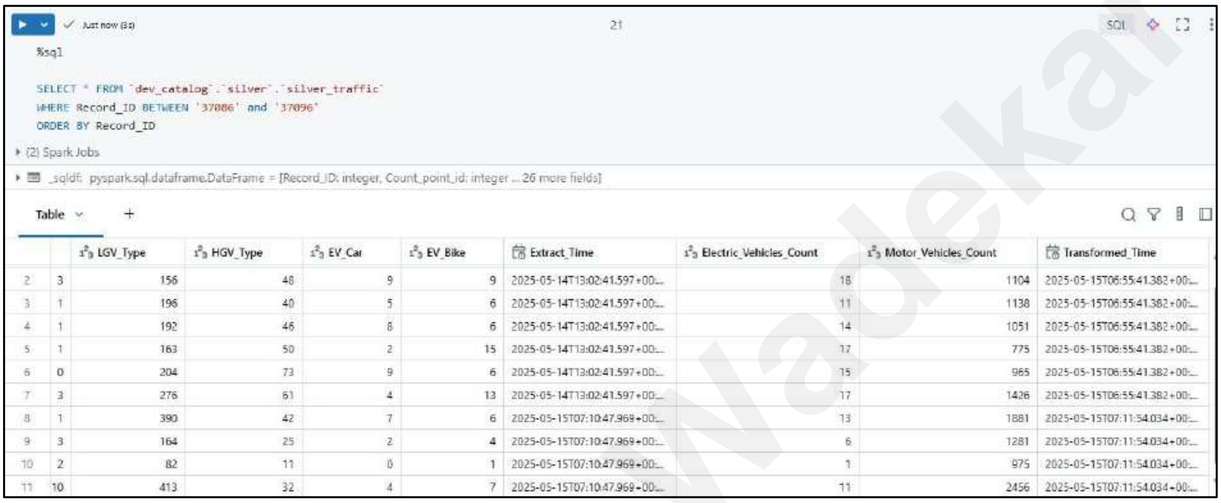
```
%sql

SELECT COUNT(*) FROM `dev_catalog`.`silver`.`silver_traffic`
```

(4) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [count(1): long]

Table

	count(1)
1	55638



Just now (3s)

```
%sql

SELECT * FROM `dev_catalog`.`silver`.`silver_traffic`
WHERE Record_ID BETWEEN '37086' and '37096'
ORDER BY Record_ID
```

(2) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [Record_ID: integer, Count_point_id: integer ... 26 more fields]

Table

		LGV_Type	HGV_Type	EV_Car	EV_Bike	Extract_Time	Electric_Vehicles_Count	Motor_Vehicles_Count	Transformed_Time
2	3	156	48	9	9	2025-05-14T13:02:41.597+00:...	18	1104	2025-05-15T06:55:41.382+00:...
3	1	196	40	5	6	2025-05-14T13:02:41.597+00:...	11	1138	2025-05-15T06:55:41.382+00:...
4	1	192	46	8	6	2025-05-14T13:02:41.597+00:...	14	1051	2025-05-15T06:55:41.382+00:...
5	1	163	50	2	15	2025-05-14T13:02:41.597+00:...	17	775	2025-05-15T06:55:41.382+00:...
6	0	204	73	9	6	2025-05-14T13:02:41.597+00:...	15	965	2025-05-15T06:55:41.382+00:...
7	3	276	61	4	13	2025-05-14T13:02:41.597+00:...	17	1426	2025-05-15T06:55:41.382+00:...
8	1	390	42	7	6	2025-05-15T07:10:47.969+00:...	13	1881	2025-05-15T07:11:54.034+00:...
9	3	164	25	2	4	2025-05-15T07:10:47.969+00:...	6	1281	2025-05-15T07:11:54.034+00:...
10	2	82	11	0	1	2025-05-15T07:10:47.969+00:...	1	975	2025-05-15T07:11:54.034+00:...
11	10	413	32	4	7	2025-05-15T07:10:47.969+00:...	11	2456	2025-05-15T07:11:54.034+00:...

Loading Data to Gold Layer

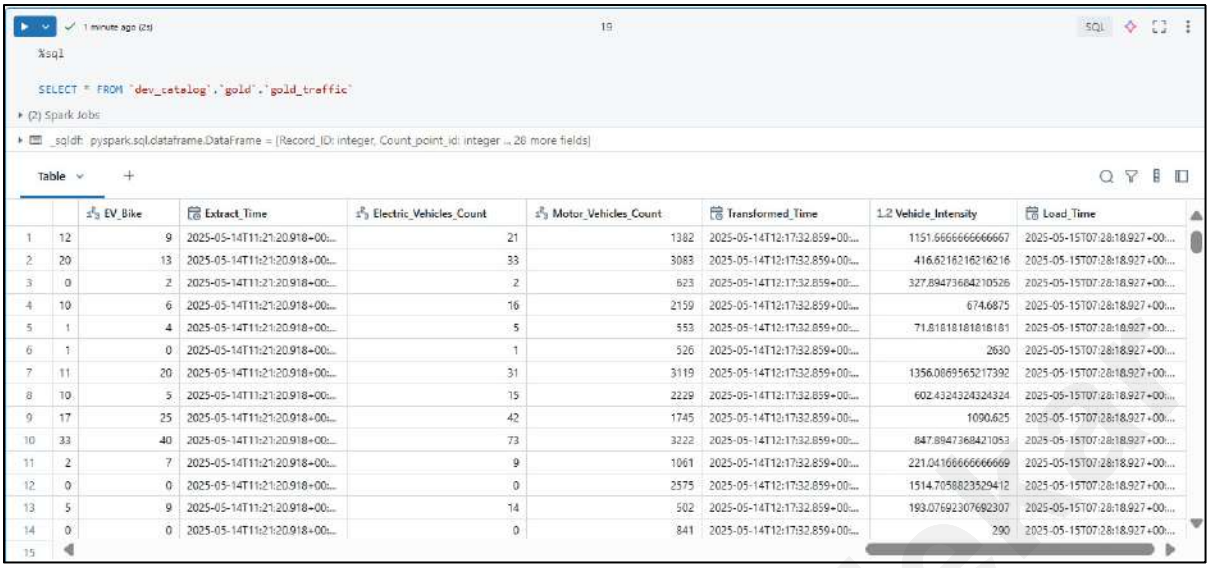
The gold layer is the final layer in the medallion architecture, designed to provide high-value datasets for reporting, dashboards, and advanced analytics. This serves as the consumption layer for PowerBI and data science use cases.

It combines enriched traffic and road data to support business insights. Optimized for performance, clarity, and ease of use by business users.

Logic for gold layer aggregations

- Created Vehicle_Itensity column in silver_traffic table - Motor_Vehicles_Count / Link_length_km
- Created Load_Time column – To check the time when the data got loaded in the table.
- Store as Delta Table/Views - Final datasets in gold layer are created as gold_traffic, gold_roads.

© Shubham Wadekar



1 minute ago (2s)

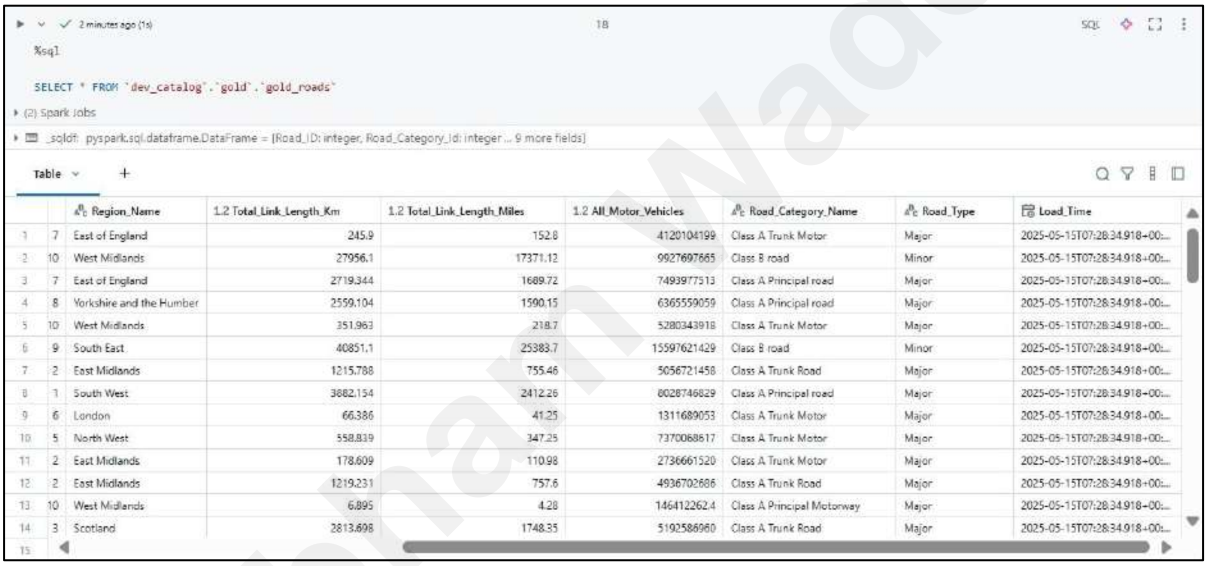
%sql

```sql
SELECT * FROM `dev_catalog`.`gold`.`gold_traffic`
```

(2) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [Record_ID: integer, Count_point_id: integer ... 28 more fields]

		EV_Bike	Extract_Time	Electric_Vehicles_Count	Motor_Vehicles_Count	Transformed_Time	Vehicle_Intensity	Load_Time
1	12	9	2025-05-14T11:21:20.918+00:...	21	1382	2025-05-14T12:17:32.859+00:...	1151.6666666666667	2025-05-15T07:28:18.927+00:...
2	20	13	2025-05-14T11:21:20.918+00:...	33	3083	2025-05-14T12:17:32.859+00:...	416.6216216216216	2025-05-15T07:28:18.927+00:...
3	0	2	2025-05-14T11:21:20.918+00:...	2	623	2025-05-14T12:17:32.859+00:...	327.89473684210526	2025-05-15T07:28:18.927+00:...
4	10	6	2025-05-14T11:21:20.918+00:...	16	2159	2025-05-14T12:17:32.859+00:...	674.6875	2025-05-15T07:28:18.927+00:...
5	1	4	2025-05-14T11:21:20.918+00:...	5	553	2025-05-14T12:17:32.859+00:...	71.81818181818181	2025-05-15T07:28:18.927+00:...
6	1	0	2025-05-14T11:21:20.918+00:...	1	526	2025-05-14T12:17:32.859+00:...	2630	2025-05-15T07:28:18.927+00:...
7	11	20	2025-05-14T11:21:20.918+00:...	31	3119	2025-05-14T12:17:32.859+00:...	1356.0869565217392	2025-05-15T07:28:18.927+00:...
8	10	5	2025-05-14T11:21:20.918+00:...	15	2229	2025-05-14T12:17:32.859+00:...	602.4324324324324	2025-05-15T07:28:18.927+00:...
9	17	25	2025-05-14T11:21:20.918+00:...	42	1745	2025-05-14T12:17:32.859+00:...	1090.625	2025-05-15T07:28:18.927+00:...
10	33	40	2025-05-14T11:21:20.918+00:...	73	3222	2025-05-14T12:17:32.859+00:...	847.8947368421053	2025-05-15T07:28:18.927+00:...
11	2	7	2025-05-14T11:21:20.918+00:...	9	1061	2025-05-14T12:17:32.859+00:...	221.04166666666669	2025-05-15T07:28:18.927+00:...
12	0	0	2025-05-14T11:21:20.918+00:...	0	2575	2025-05-14T12:17:32.859+00:...	1514.7058823529412	2025-05-15T07:28:18.927+00:...
13	5	9	2025-05-14T11:21:20.918+00:...	14	502	2025-05-14T12:17:32.859+00:...	193.07692307692307	2025-05-15T07:28:18.927+00:...
14	0	0	2025-05-14T11:21:20.918+00:...	0	841	2025-05-14T12:17:32.859+00:...	290	2025-05-15T07:28:18.927+00:...



2 minutes ago (1s)

%sql

```sql
SELECT * FROM `dev_catalog`.`gold`.`gold_roads`
```

(2) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [Road_ID: integer, Road_Category_Id: integer ... 9 more fields]

		Region_Name	Total_Link_Length_Km	Total_Link_Length_Miles	All_Motor_Vehicles	Road_Category_Name	Road_Type	Load_Time
1	7	East of England	245.9	152.8	4120104199	Class A Trunk Motor	Major	2025-05-15T07:28:34.918+00:...
2	10	West Midlands	27956.1	17371.12	9927697665	Class B road	Minor	2025-05-15T07:28:34.918+00:...
3	7	East of England	2719.344	1689.72	7493977513	Class A Principal road	Major	2025-05-15T07:28:34.918+00:...
4	8	Yorkshire and the Humber	2559.104	1590.15	6365559059	Class A Principal road	Major	2025-05-15T07:28:34.918+00:...
5	10	West Midlands	351.963	218.7	5280343918	Class A Trunk Motor	Major	2025-05-15T07:28:34.918+00:...
6	9	South East	40851.1	25383.7	15597621429	Class B road	Minor	2025-05-15T07:28:34.918+00:...
7	2	East Midlands	1215.788	755.46	5056721458	Class A Trunk Road	Major	2025-05-15T07:28:34.918+00:...
8	1	South West	3882.154	2412.26	8028746829	Class A Principal road	Major	2025-05-15T07:28:34.918+00:...
9	6	London	66.386	41.25	1311689053	Class A Trunk Motor	Major	2025-05-15T07:28:34.918+00:...
10	5	North West	558.839	347.25	7370068617	Class A Trunk Motor	Major	2025-05-15T07:28:34.918+00:...
11	2	East Midlands	178.609	110.98	2736661520	Class A Trunk Motor	Major	2025-05-15T07:28:34.918+00:...
12	2	East Midlands	1219.231	757.6	4936702686	Class A Trunk Road	Major	2025-05-15T07:28:34.918+00:...
13	10	West Midlands	6.895	4.28	146412262.4	Class A Principal Motorway	Major	2025-05-15T07:28:34.918+00:...
14	3	Scotland	2813.698	1748.35	5192586960	Class A Trunk Road	Major	2025-05-15T07:28:34.918+00:...

Jobs to orchestrate the flow

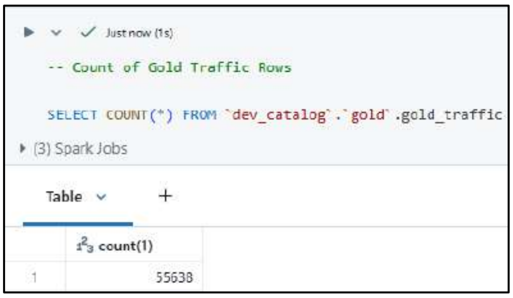
- There are certain notebooks which need to run daily to get the data, and some notebooks need not run daily. Here also we need to load data to bronze table, silver layer transformations and the gold transformations. So, all these notebooks need to be executed one after another. Based on the cadence when the data arrives, we need to run these notebooks in a flow.

- So, to run these notebooks in a flow, we need to give a job. A job will orchestrate all these notebooks, and it will run all these notebooks in a flow.

- Created a playground notebook that has the count of all the records. Counts in gold layer for both datasets:

© Shubham Wadekar

- gold_traffic: 55638

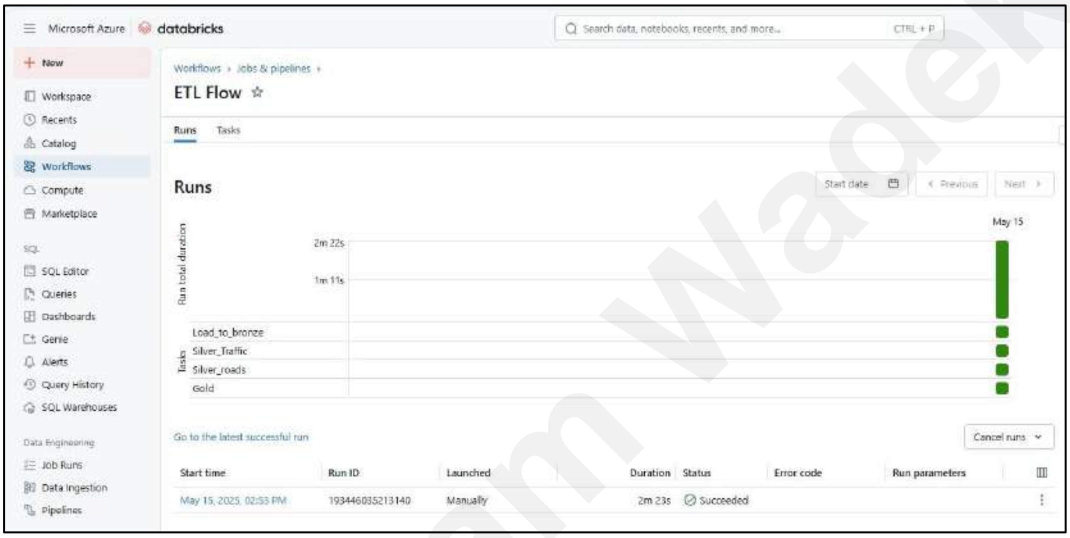


- gold_roads: 76

- Created a job named ETL Flow having various tasks:
  - Task Name: Load_to_Bronze
    Cluster Used: Job Cluster (Once completed, it terminates)
  - Task Name: Silver_Traffic
  - Task Name: Silver_Roads
  - Task_Name: Gold

- We added new csv files in the landing zone for raw_traffic and raw_roads. Now we just need to run this ETL Flow workflow and check the counts. The new records will be added.

© Shubham Wadekar

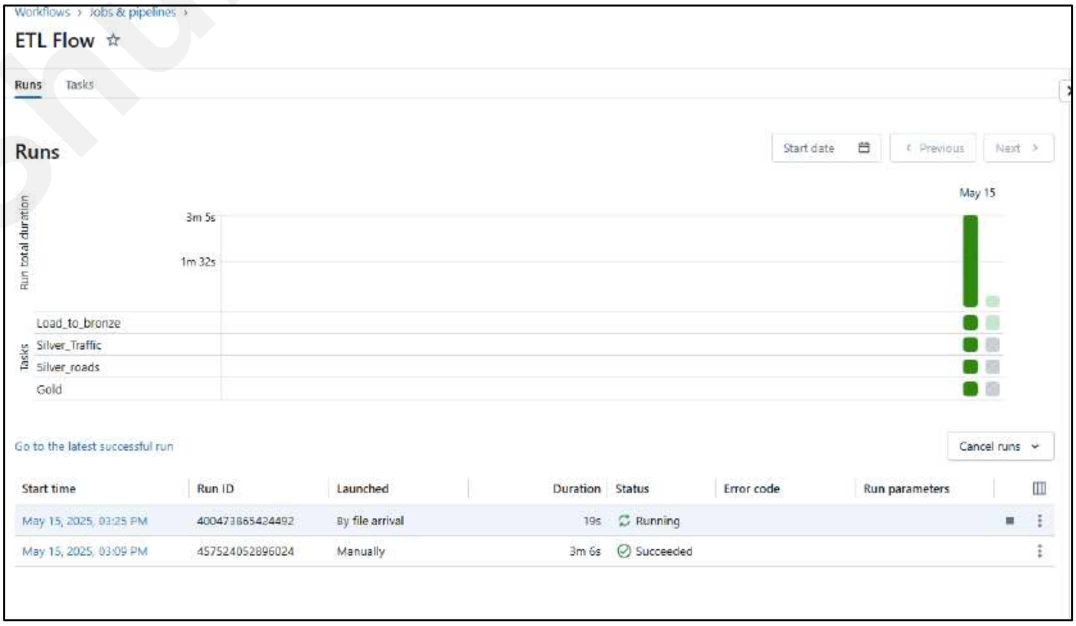


- The counts changed after running the ETL Flow.

© Shubham Wadekar

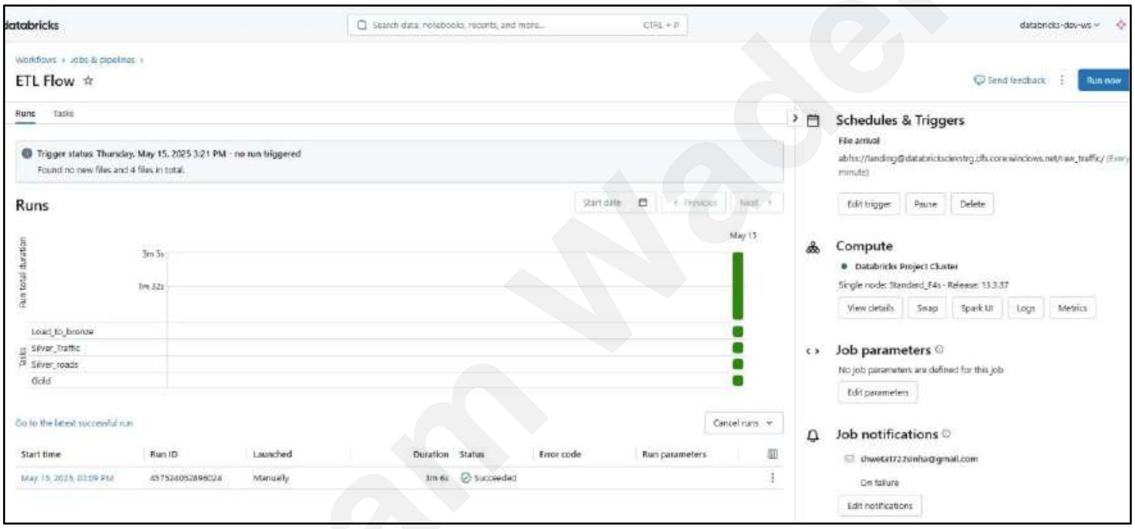
- Added Triggers:
  - Raw_Traffic will be updated twice a day, not sure for the time so we use **file arrival** trigger here. (tracking for new file path – landing external location/raw_traffic/)

After every one minute it checks for the file as we have the file arrival trigger available. When we upload a new csv file for raw_traffic in the landing zone it will automatically run the ETL Flow job. In the Launched section we can see the job started running by the file arrival.

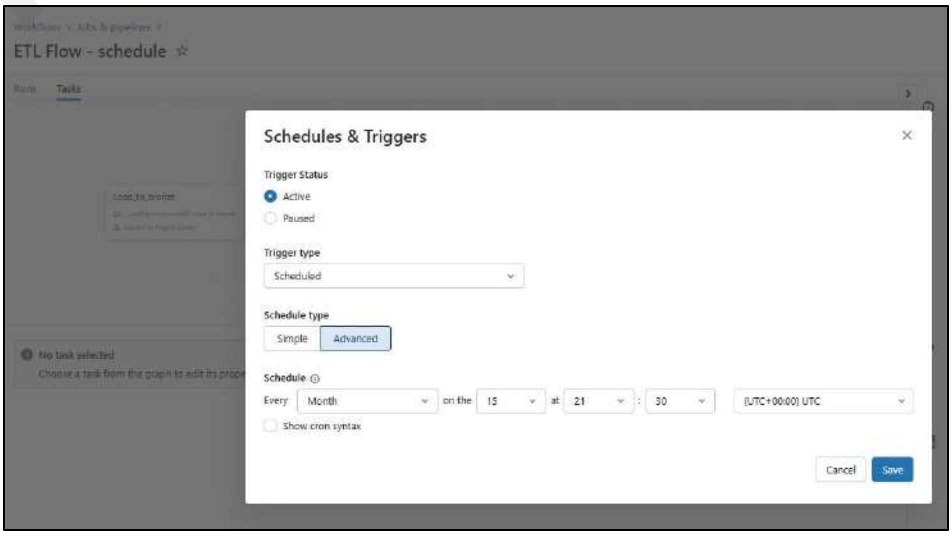


© Shubham Wadekar

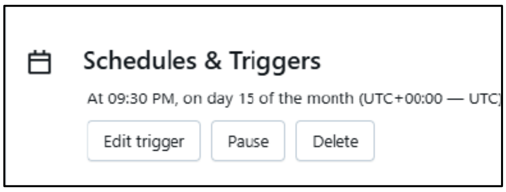
Added notification email on failure.



- Raw_Roads will be updated monthly basis, because this is kind of fixed (example – road length, type of road). We cloned the ETL Flow workflow and edited the trigger to scheduled type for every month at a particular time, because we cannot add 2 triggers within the same workflow.



© Shubham Wadekar

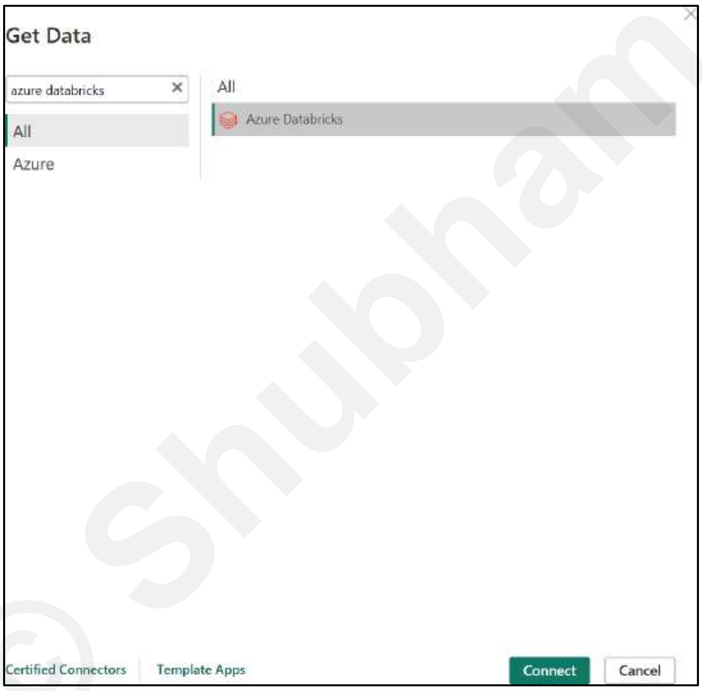


Reporting Data to PowerBI

PowerBI is used as the reporting and visualization tool to consume and present the Gold Layer data stored in Azure Data Lake via Azure Databricks. It provides interactive dashboards and data-driven insights from traffic and roads data. This will help to support decision-makers in analyzing traffic patterns, road utilization, and vehicle trends across different regions and timeframes.

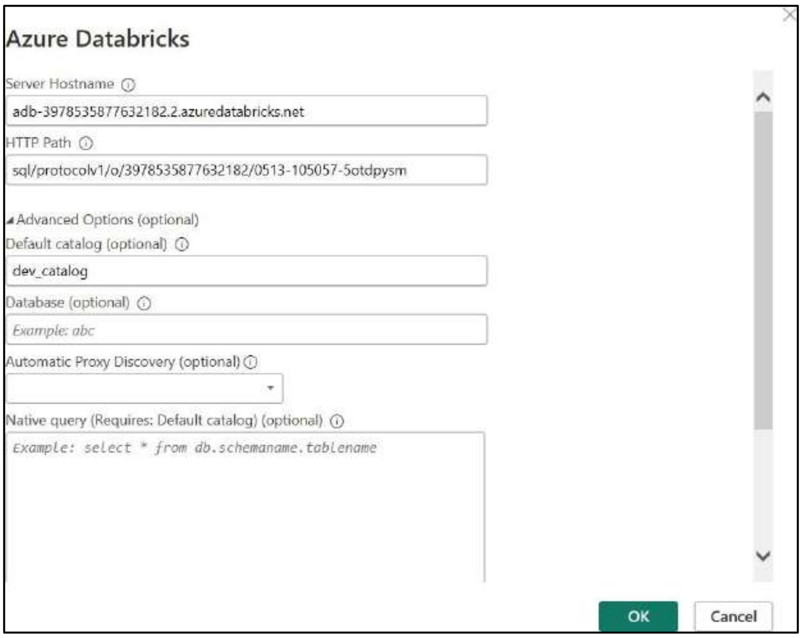
PowerBI connection with Azure Databricks

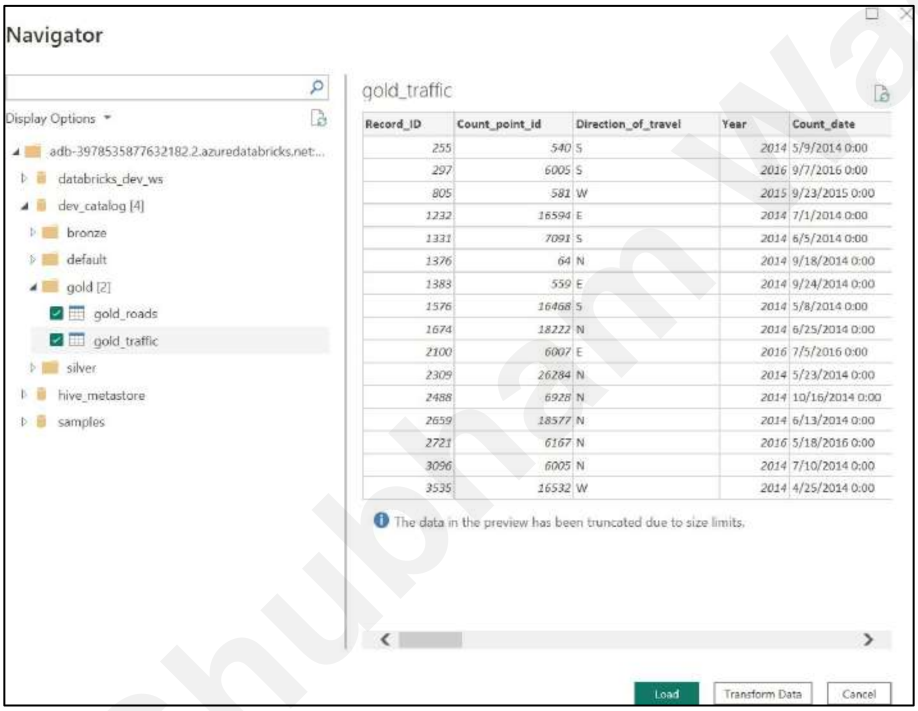
To get the gold data, select the 'Get Data' in PowerBI and search for Azure Databricks. PowerBI connects to the Gold Layer Delta tables (gold_traffic, gold_roads).



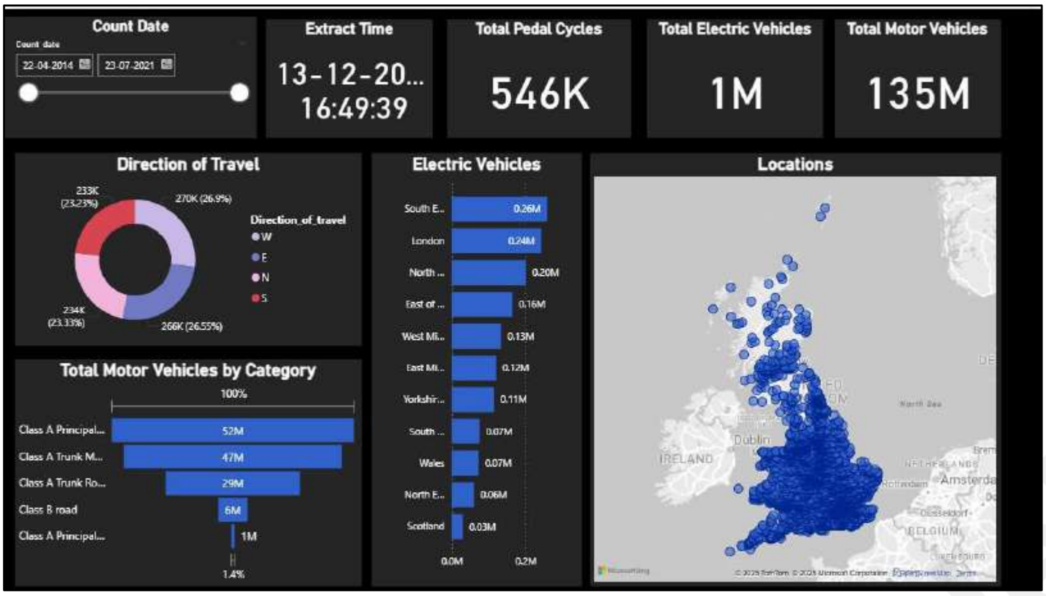
Using the Databricks compute details we connect PowerBI to Azure Databricks as shown below.

© Shubham Wadekar





© Shubham Wadekar

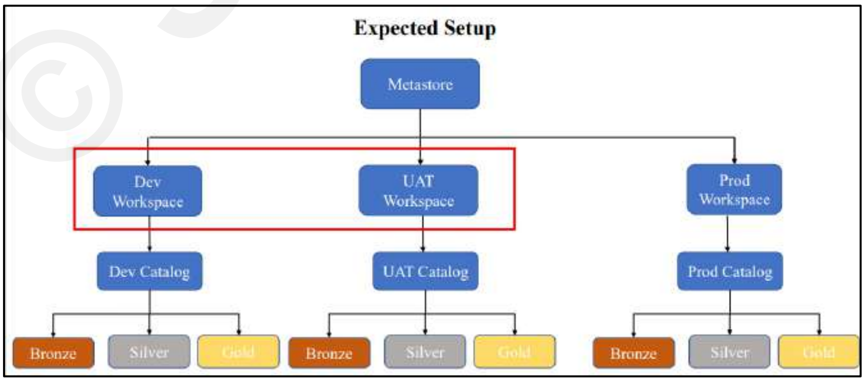


This PowerBI dashboard presents key insights from the gold layer of the project using curated traffic and road data.

Key points of the dashboard:

- Date filters: Users can filter data by count date and view when the data was last ingested.
- Extract Time: Shows when the dashboard is refreshed.
- KPI Tiles: Show total counts of:
  - Pedal Cycles
  - Electric Vehicles
  - Motor Vehicles
- Direction of Travel (Donut Chart): Shows vehicle distribution by travel direction (N, S, E, W).
- Electric Vehicles by Region (Bar Chart): Highlights regional EV usage, with Southeast and London leading.
- Motor Vehicles by Road Category: Displays total vehicle count across road types (e.g., Class A Principal, Class B).
- Location Map: Geospatial view of all traffic count points across the UK.

<u>Continuous Integration & Continuous Deployment (CI/CD)</u>



© Shubham Wadekar

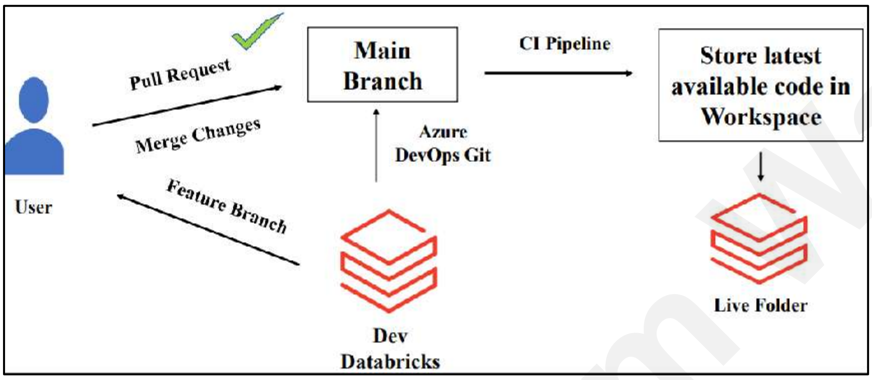
In this project, we have used Azure DevOps git as our repository. We have created the UAT workspace for testing and we have implemented the CI/CD. This project uses Azure DevOps to implement a robust Continuous Integration and Continuous Deployment (CI/CD) pipeline for managing notebooks, workflows, and configurations in Databricks. To ensure that code updates are version-controlled and tested before deployment.

Since this is a sample project, we have created the UAT workspace and we have implemented the CI/CD, where we have all the data from the dev workspace to the UAT workspace.

We have created the catalog and dynamically created all the schemas to represent the medallion architecture.

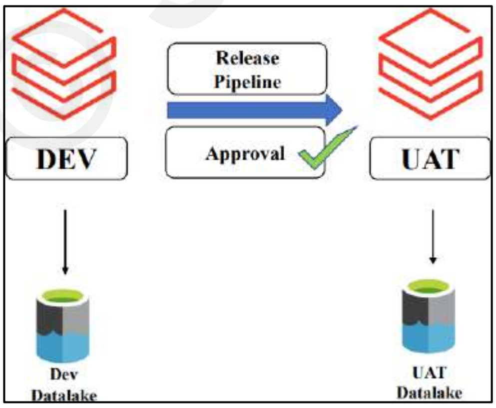
Data from dev   Data to UAT

Continuous Integration



- Git repository used: Azure DevOps git
- Main branch holds all the changes done to the project. Whatever we work on notebooks, it is stored in a centralized place that is called the main branch.
- Continuous Integration lets multiple developers work and all the changes are merged with the main branch repository.
- Pull requests need to be approved by a technical lead or technical head in the team, who has access to Azure DevOps.

Continuous Deployment

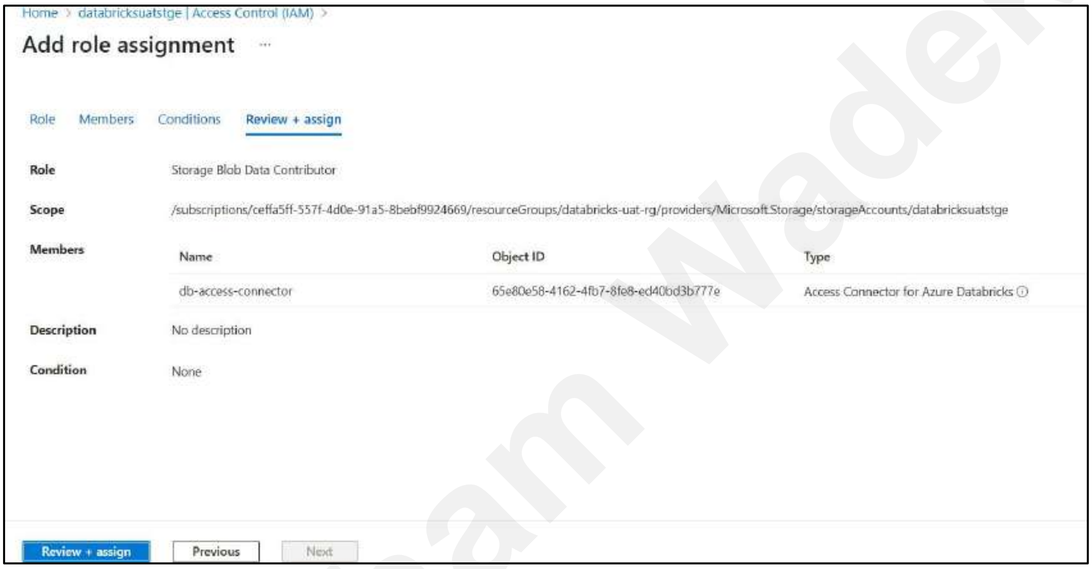


© Shubham Wadekar

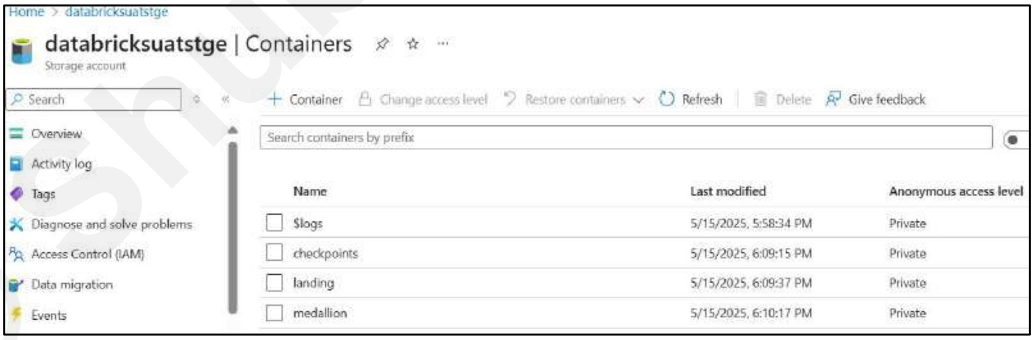
Release Pipeline: It gets all the changes in a live folder to UAT workspace. This will go on after an approval system has been done.

UAT Resources

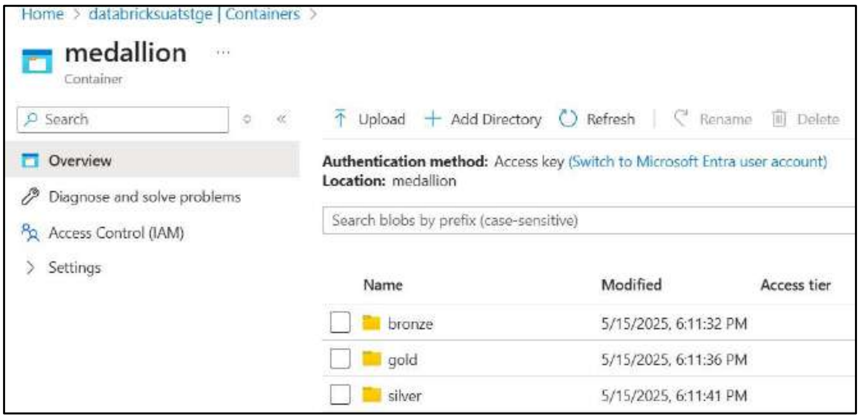
- Resource Group: databricks-uat-rg
- Databricks workspace: databricks-uat-wsp
- Storage account: databricksuatstge
- Provided role assignment as storage blob data contributor to db-access-connector (Access connector for Azure Databricks) and gave the managed identity. Now UAT workspace will be a part of Unity Catalog.



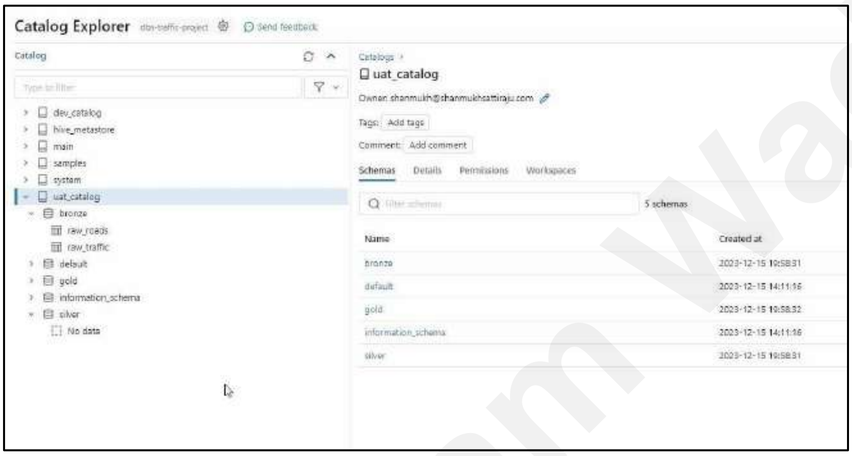
- Created the same containers as we created for the dev workspace.



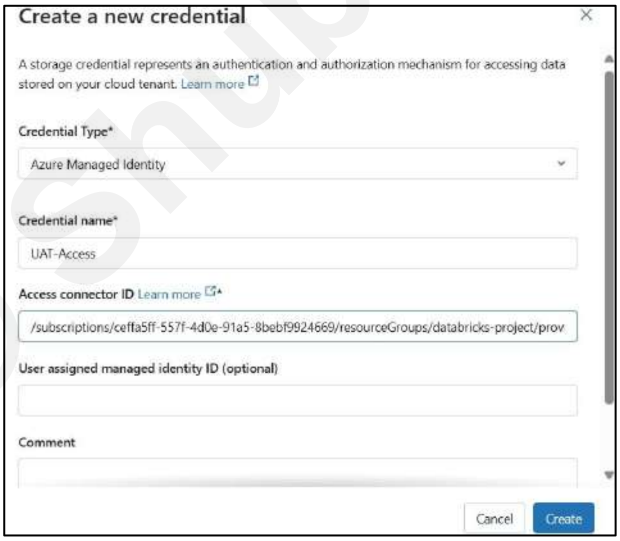
© Shubham Wadekar



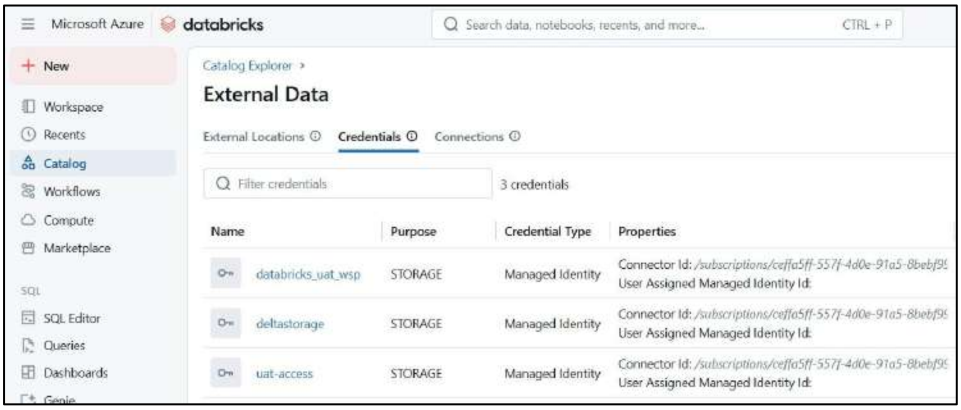
- Created uat_catalog with all the schemas.



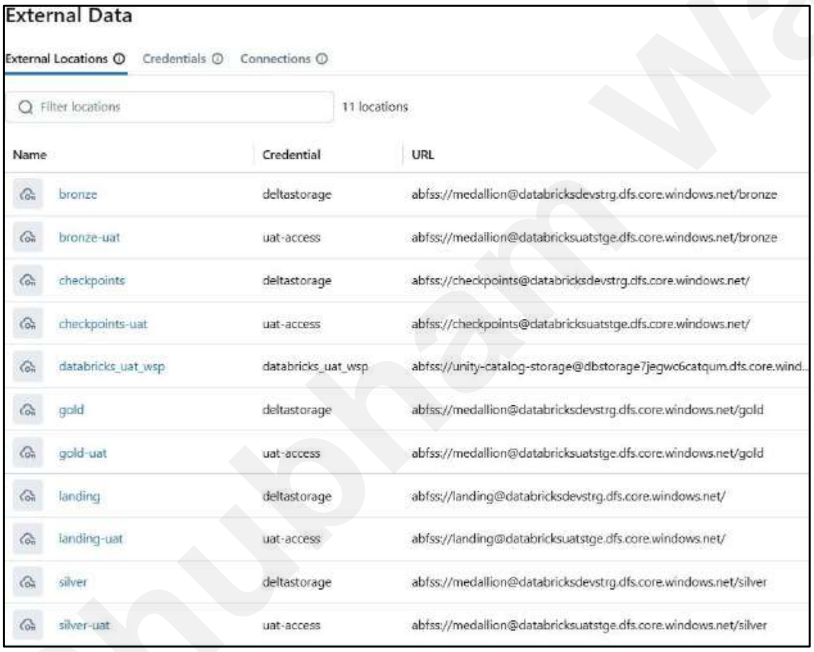
- Created another credential for creating external locations.



© Shubham Wadekar



- Created 5 external locations for UAT also
  - bronze-uat
  - silver-uat
  - gold-uat
  - checkpoints-uat
  - landing-uat

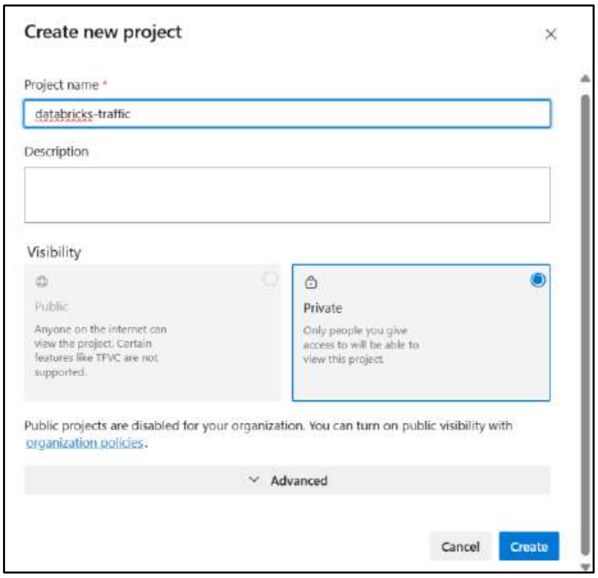


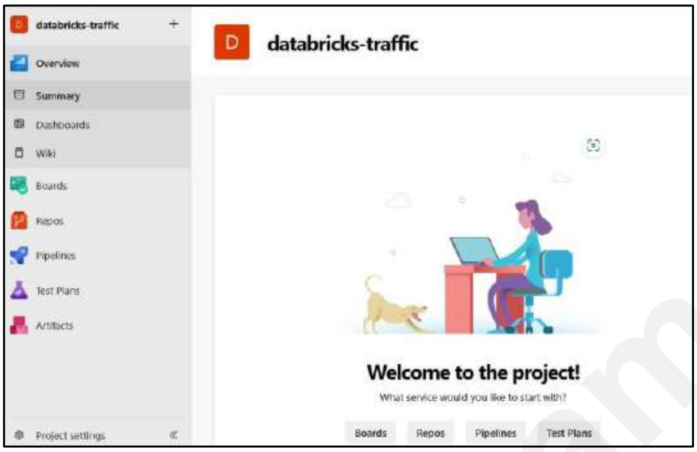
<u>Integrating Azure DevOps with Azure Databricks</u>

Continuous Integration:

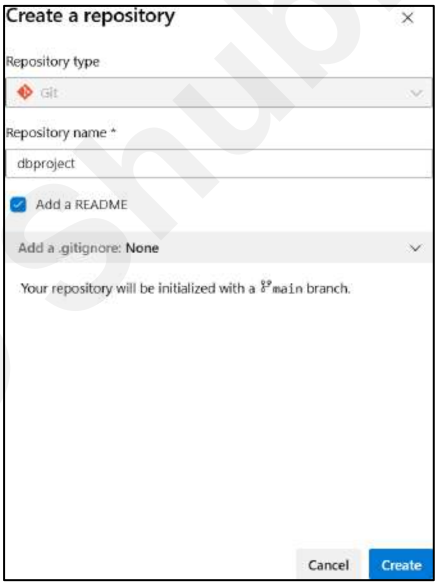
Created a new project in Azure DevOps named as databricks-traffic.

© Shubham Wadekar



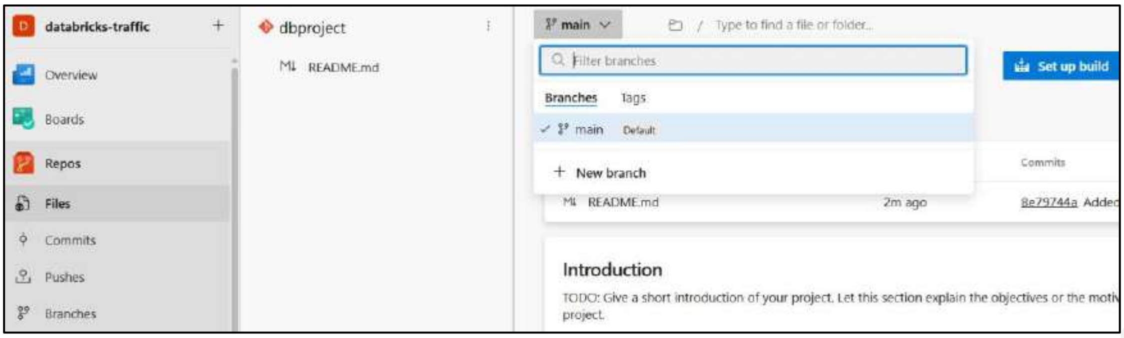


- Created a repository in Azure DevOps.

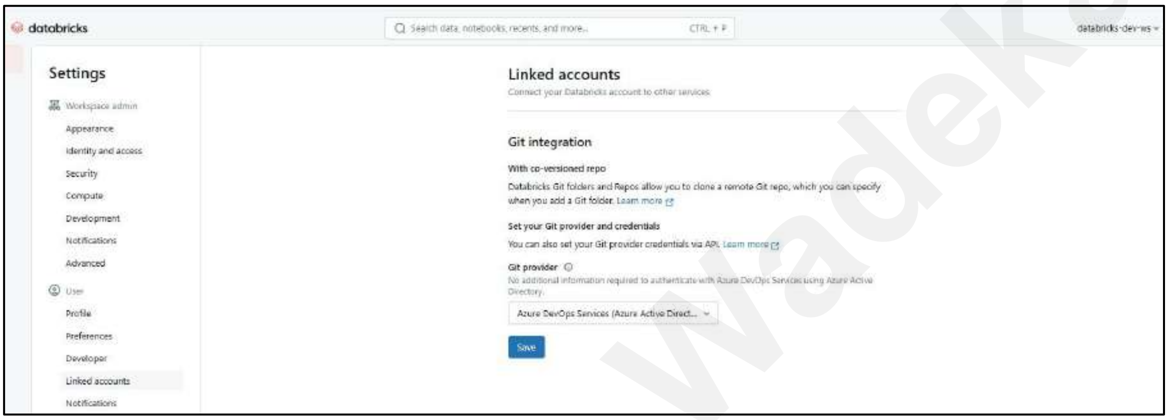


We can see the main branch in our repository (dbproject) as shown below. This is the place where every code will be copied. This is used as the central repository.

© Shubham Wadekar



- Link your Azure databricks dev workspace to Azure DevOps Services (Azure Active

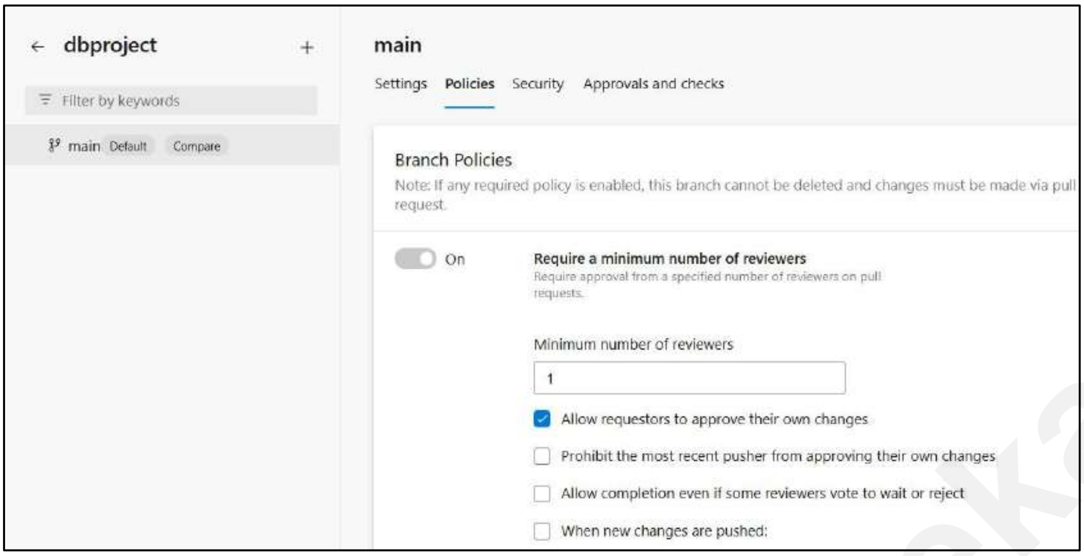


Directory) in User Settings.

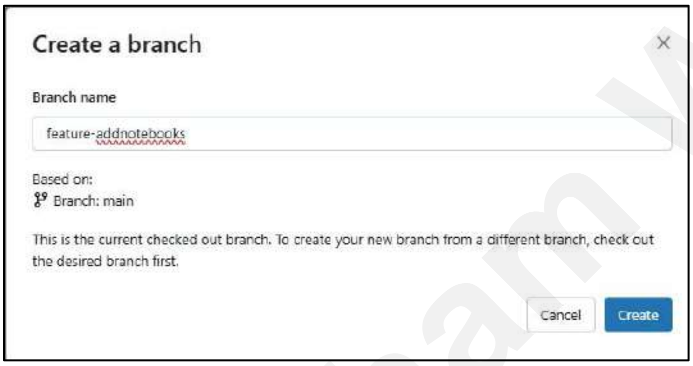
- Created our own repository in Azure Databricks so that we can integrate Azure Databricks with Azure DevOps. We cloned our new project created in Azure DevOps and copied the http link and pasted while creating a Repo to get all its details.

- In Azure DevOps we have set the minimum number of reviewers to be 1. So, when we have a pull request, it needs to be approved by any technical person in this project which I am doing. Mostly in every project we have more than 1 so we would be needing more people to approve then. Since I am the only one working here so I will approve my own pull request here.

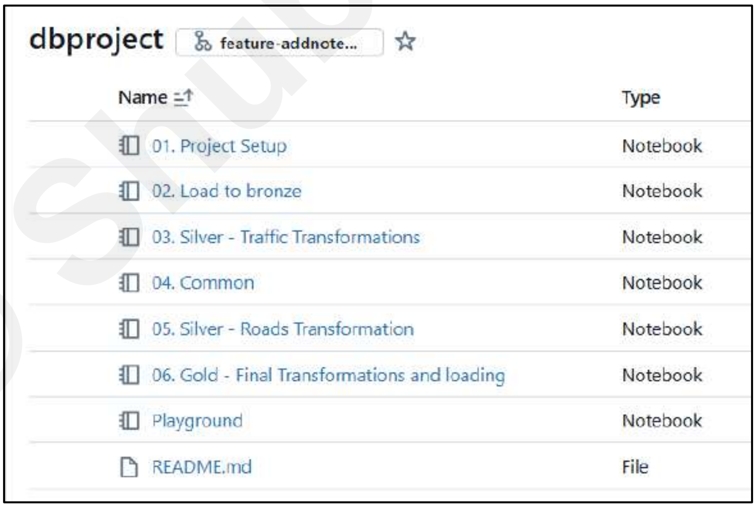
© Shubham Wadekar



- We can see we have a main branch and currently we can see it is empty. We need to have all our codes in the main branch, and for that we have created a feature branch in which we will create a pull request.



- After creating our feature branch, we moved all the codes from the user's workspace to

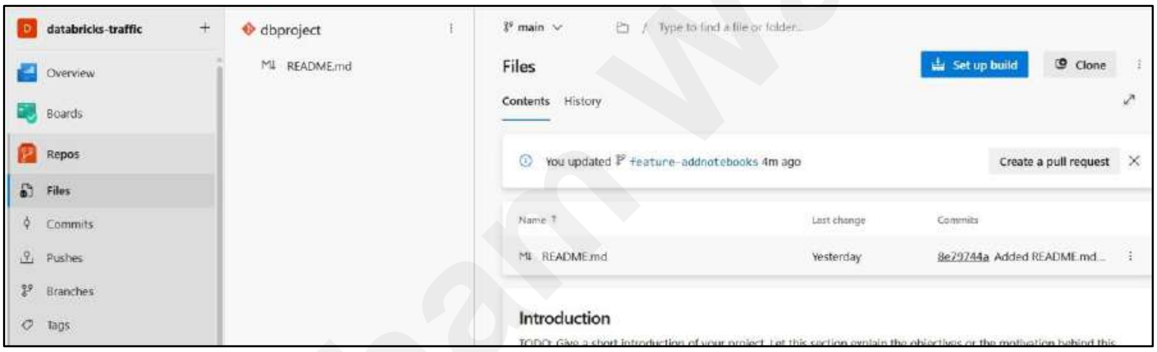


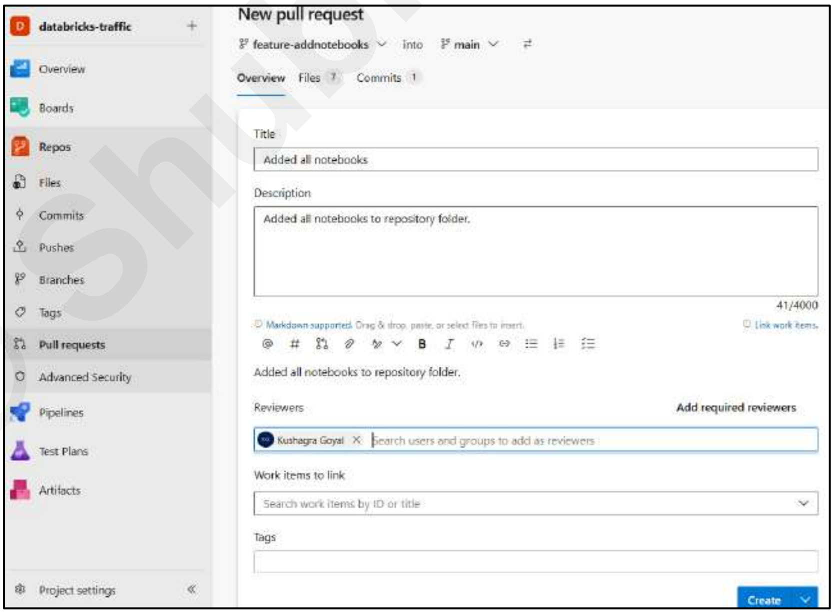
our own repository in our feature branch (feature-addnotebooks) for this project.

- We need to save all these changes to our feature branch. So, we will commit and push the changes to our feature branch as shown below.

© Shubham Wadekar

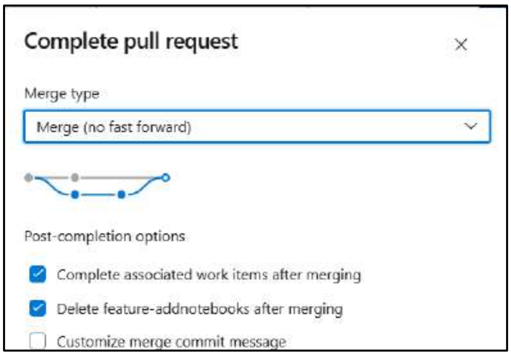
- Since we did the commit and push to our feature branch, in Azure DevOps we will get notified in our main branch that a new branch has been created that is having some commit. Based on that commit we will be creating the pull request.





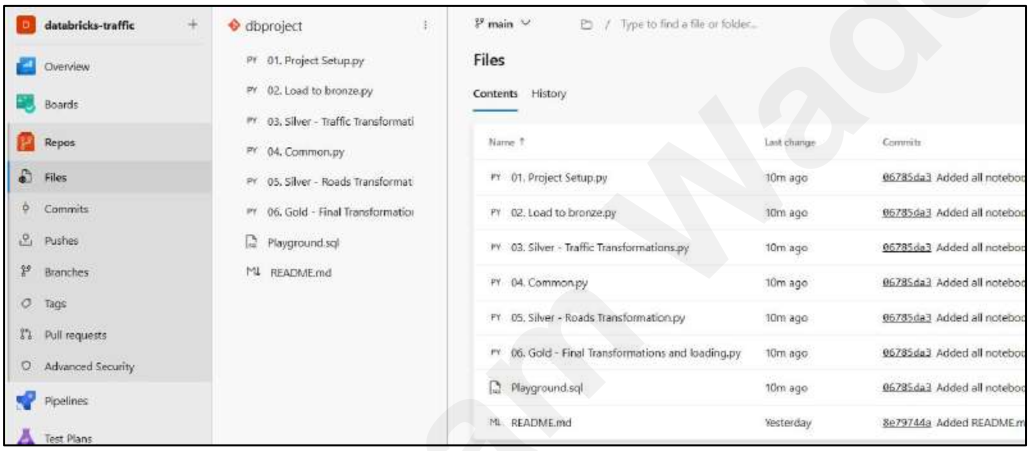
- Now we need to create the pull request.

© Shubham Wadekar

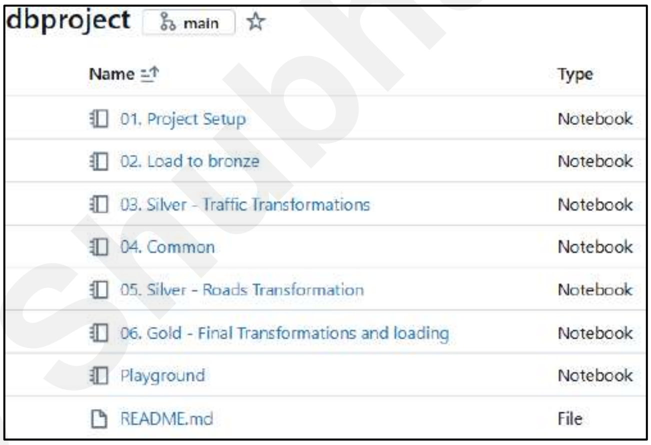


- Approved the pull request and completed it.

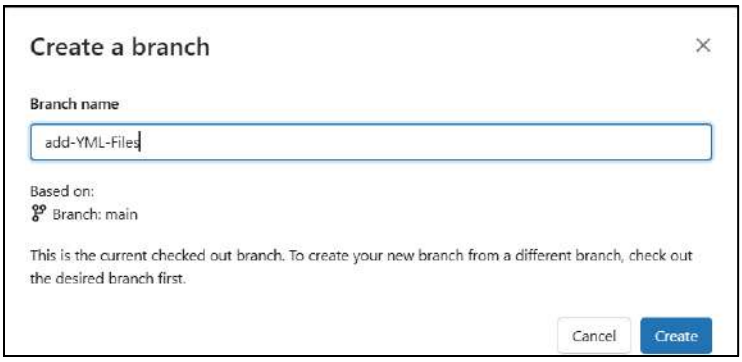
- Now we can see that all our codes are available in the main branch in Azure DevOps and in



Azure Databricks as well.

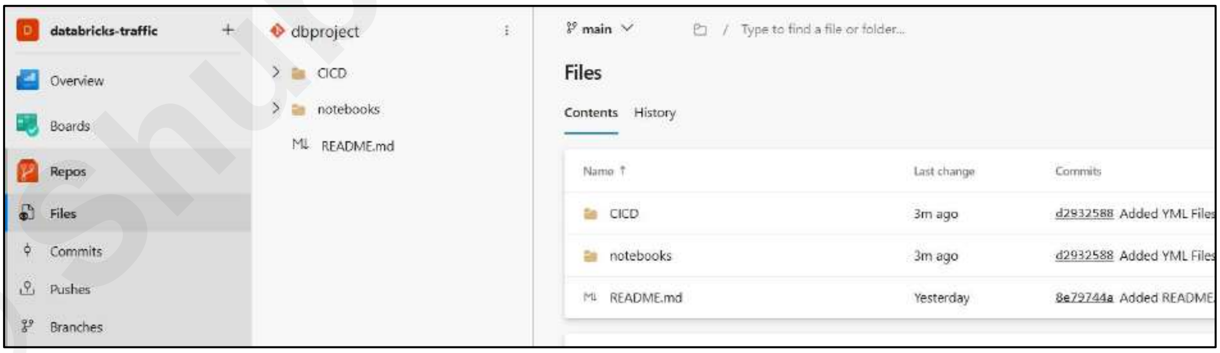


© Shubham Wadekar



- Created a feature branch to add some YML files.

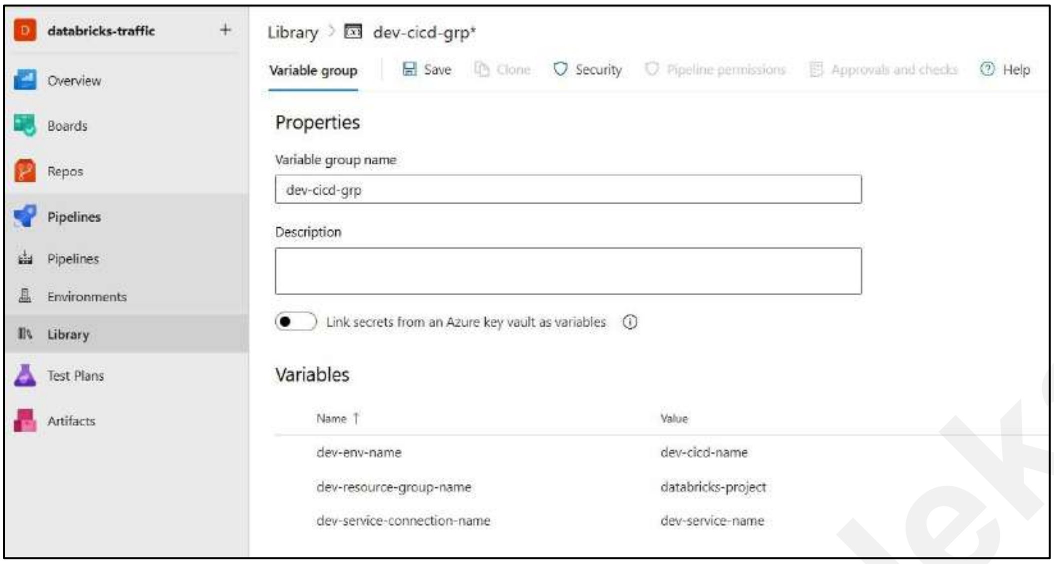
- Made separate folders for CICD and all notebooks. Committed and pushed the changes to the feature branch and created and completed a pull request in Azure DevOps.

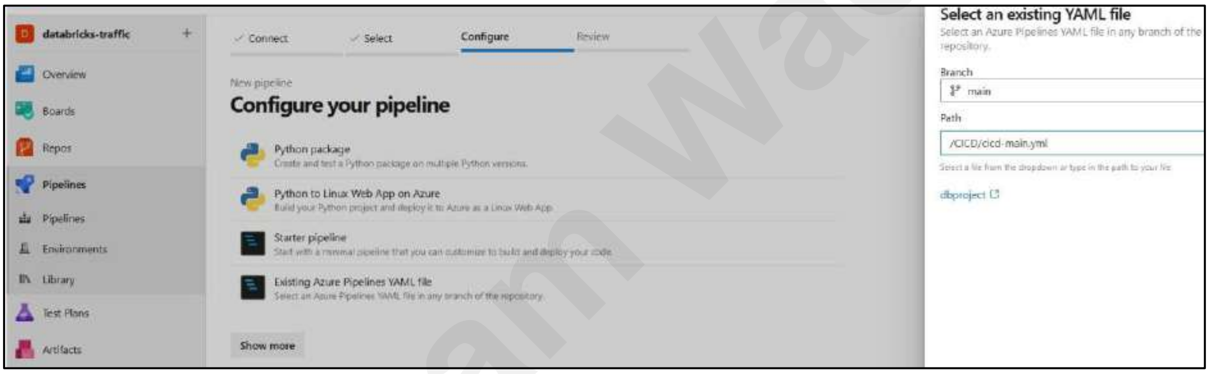


- The CICD YML File checks that when there is a change in the main branch, it needs to trigger the CI pipeline, and it needs to deploy notebooks in the live folder.

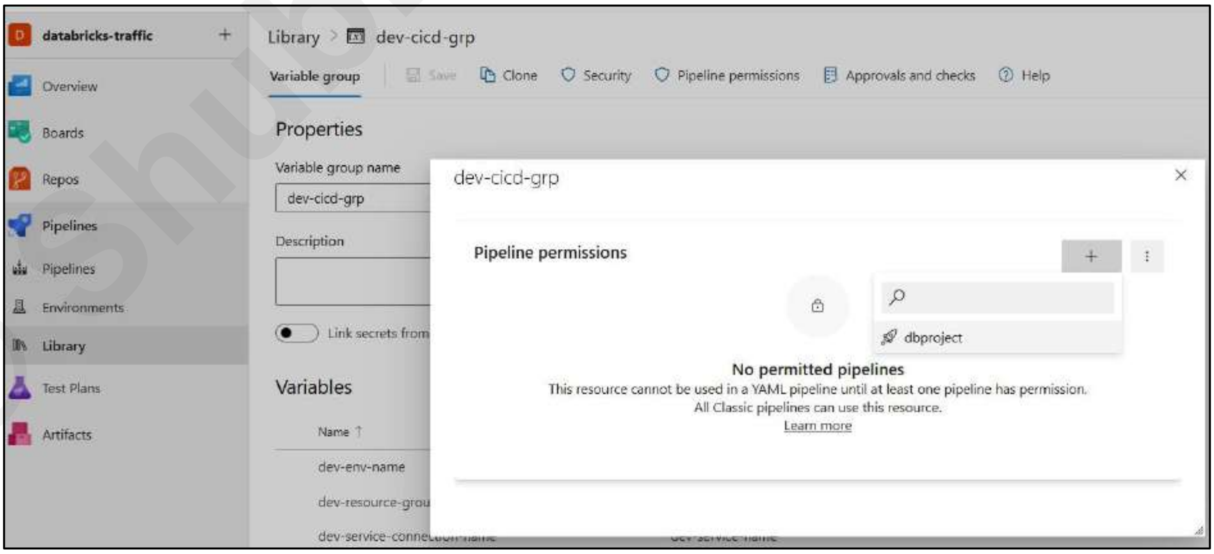
© Shubham Wadekar

- Created a library (dev-cicd-grp) and added all the required variables in the library.





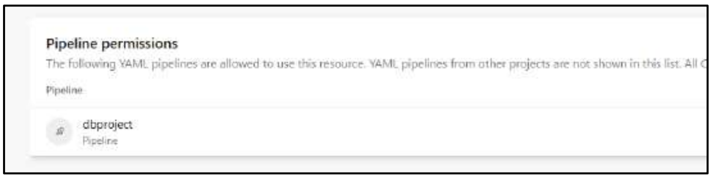
- Created a pipeline and saved it.



- Gave the permission to the library that we created with the new pipeline.

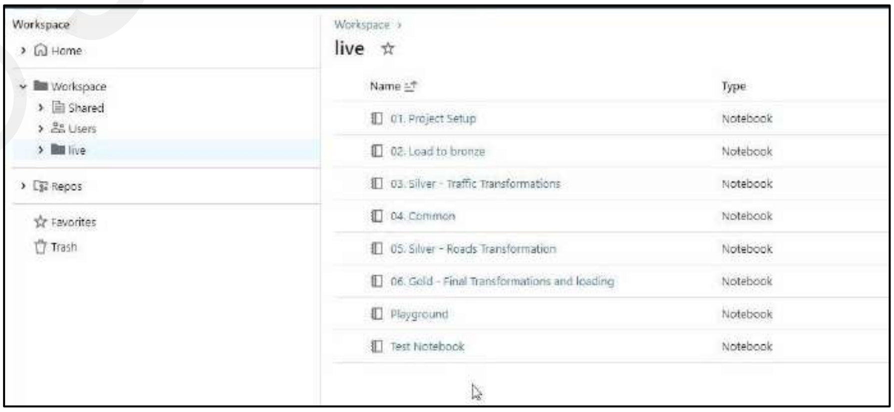
© Shubham Wadekar

- Added pipeline permission to the environment and service connection in their security option.



- To test the pipeline that we created in Azure DevOps we created a test notebook in our feature branch and merged it with the main branch. We can see that the pipeline will start running on its own as soon as there is any change in the main branch.

- After running the pipeline, we can see that there is a live folder created in Azure Databricks which has all the notebooks. So, every time whoever pushes their changes to main branch all the codes will be copied to the live folder.

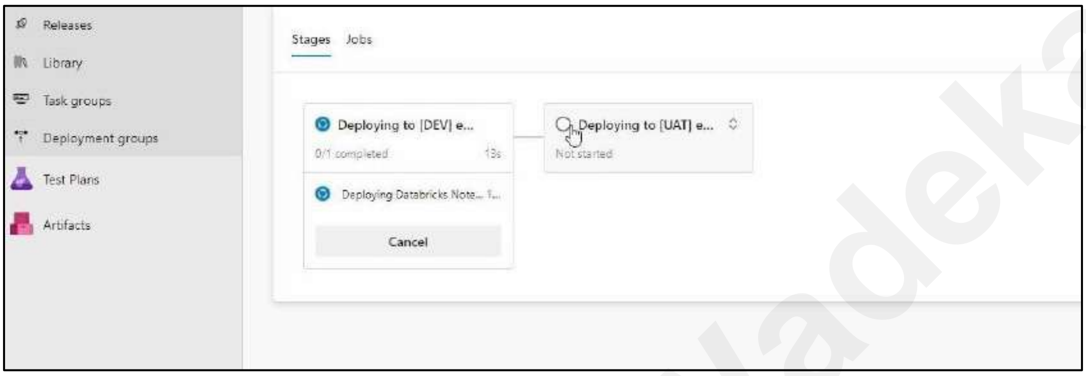


© Shubham Wadekar

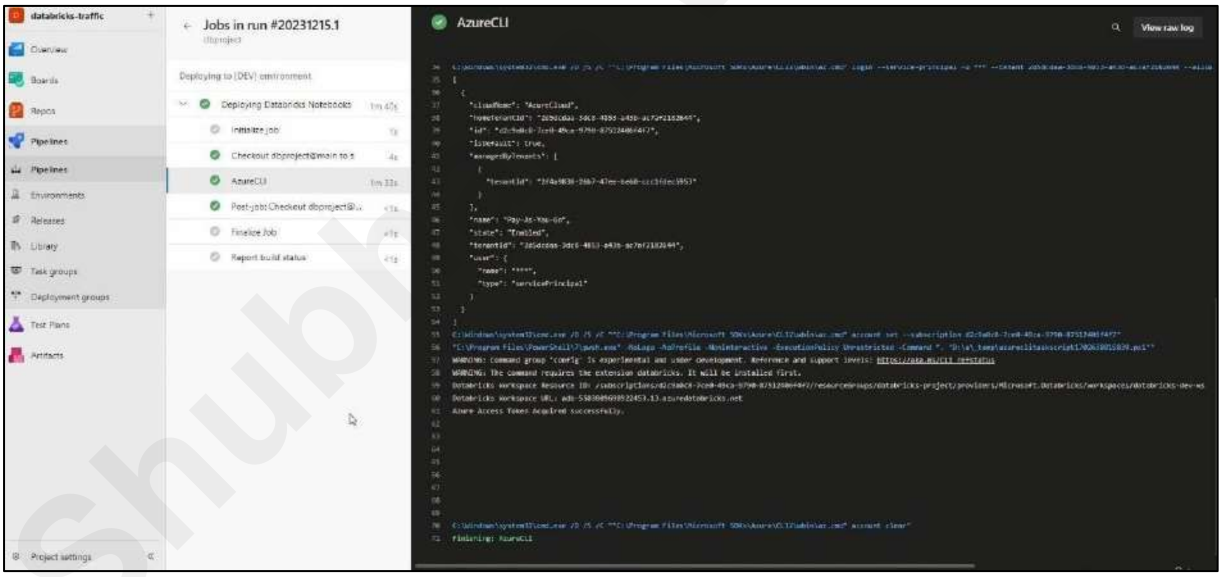
Continuous Deployment:

- Previously we added a group where we need to get the credentials of all dev workspaces. Since we want to deploy this to UAT we need to create a group that would hold all the variables of UAT environment. Then create variables for the UAT environment just like we did for dev.

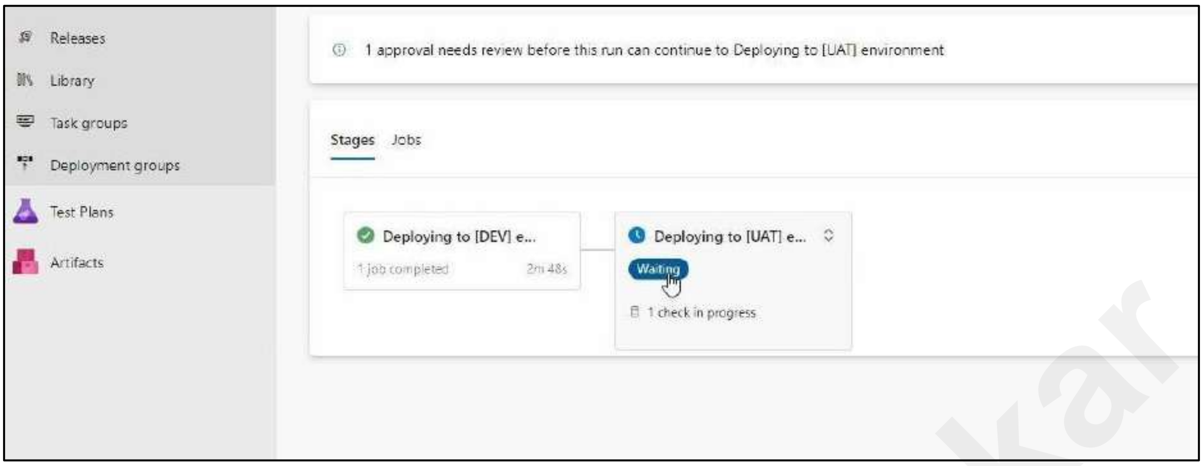
- Tested with a change we can see that the main CI pipeline started running and it got



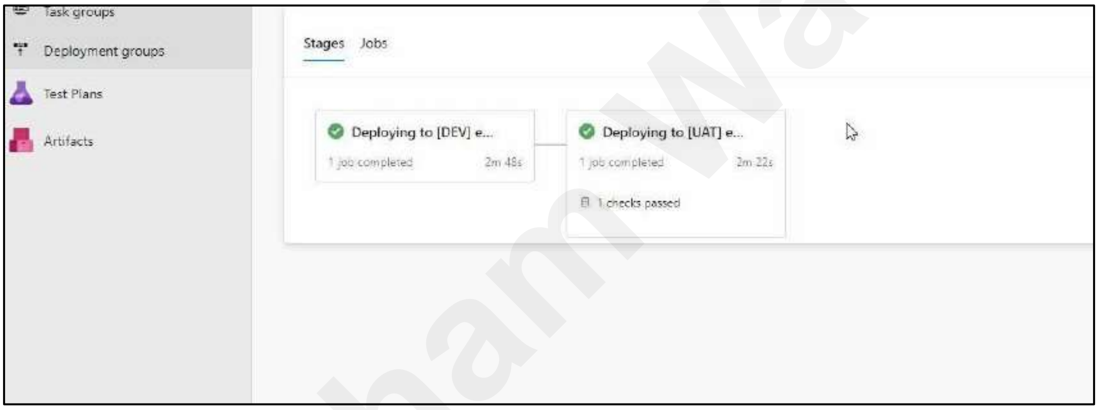
deployed to the dev environment.

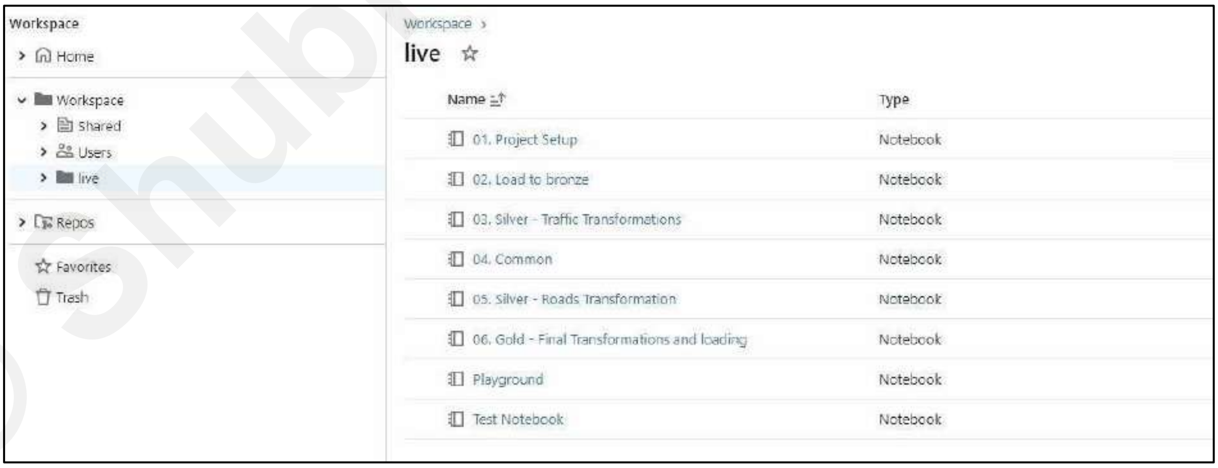


- After deploying it to dev, it is waiting for the approval so that it can deployed to the UAT environment. We can go and confirm this and give the approval of the required user.



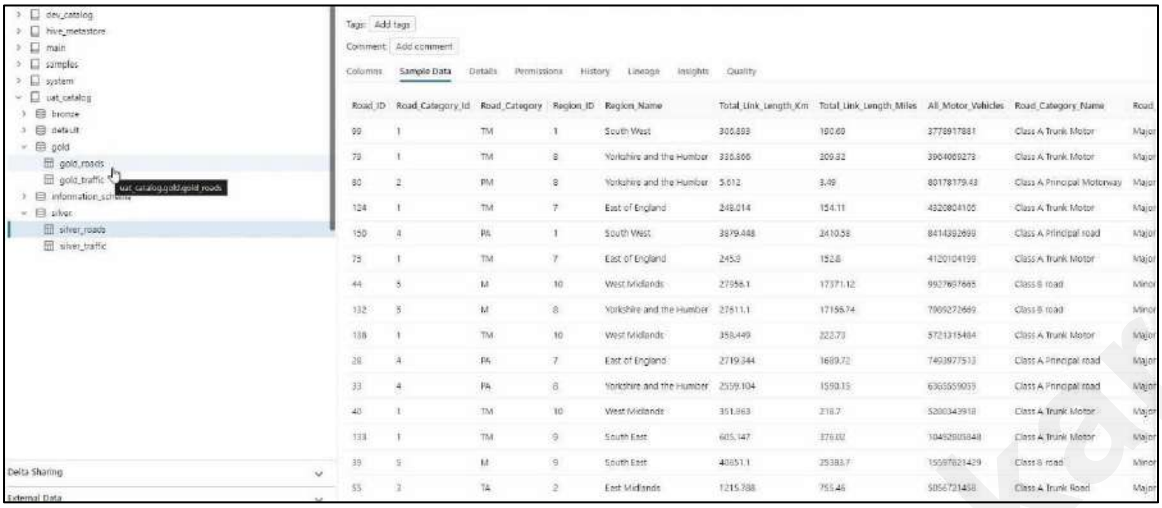
- After deploying it to UAT we can see the live folder in Azure Databricks with all the codes in our UAT workspace also.





- After running all the notebooks, we can get all the data in UAT environment.

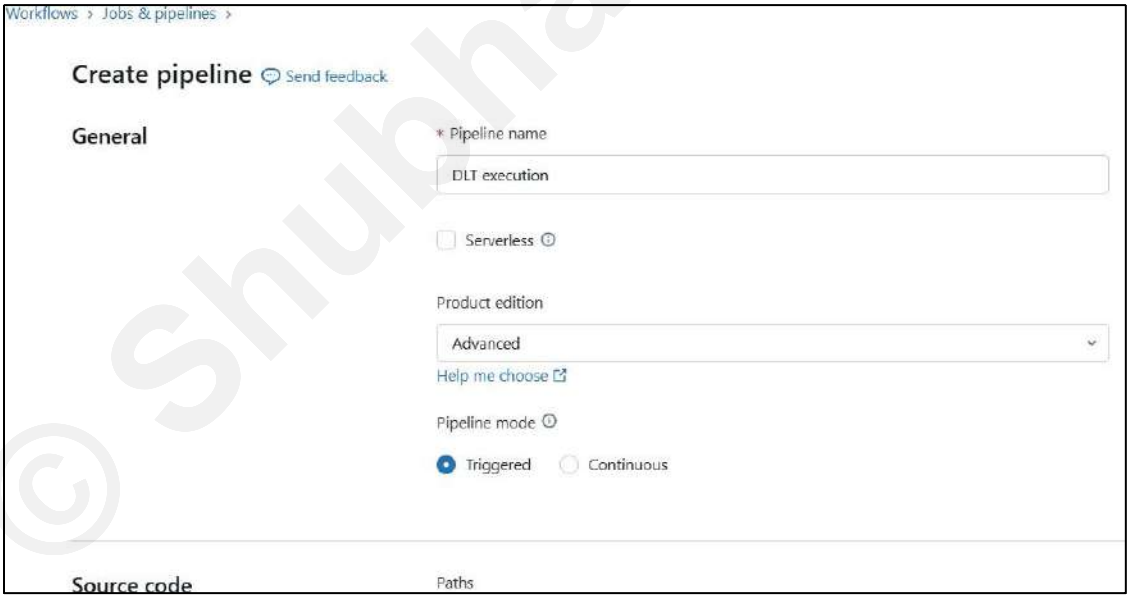
© Shubham Wadekar



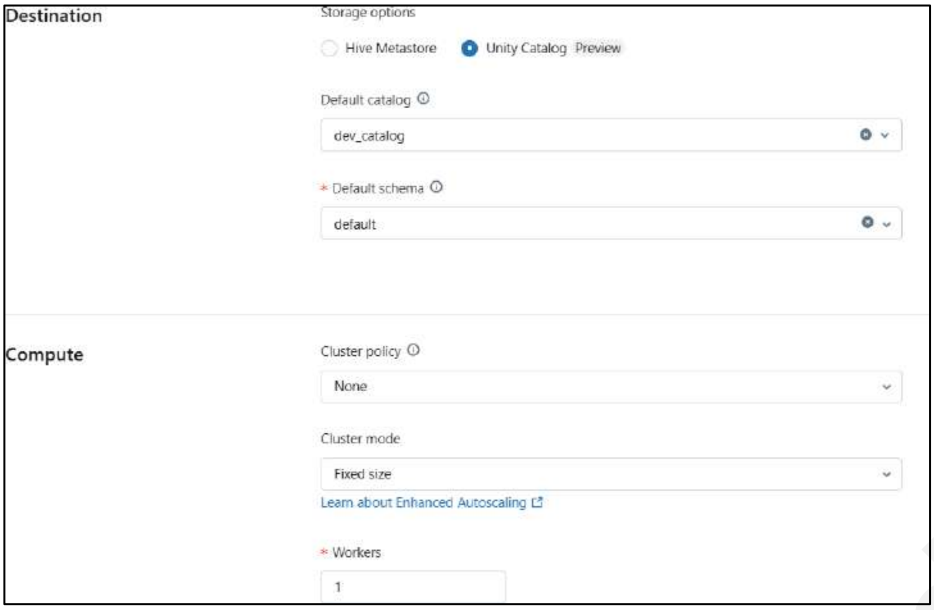
Delta Live Table

Delta Live Tables (DLT) is used in this project to orchestrate, automate, and manage the data pipeline from raw data ingestion to the creation of gold-level analytics tables. It automates the creation of bronze, silver, and gold tables with built-in data quality checks and lineage tracking. It reduces the complexity of manual job scheduling and notebook chaining.

- Created DLT in default schema.
- Used autoloader to handle incremental loading.
- Mode of pipeline: Triggered



© Shubham Wadekar

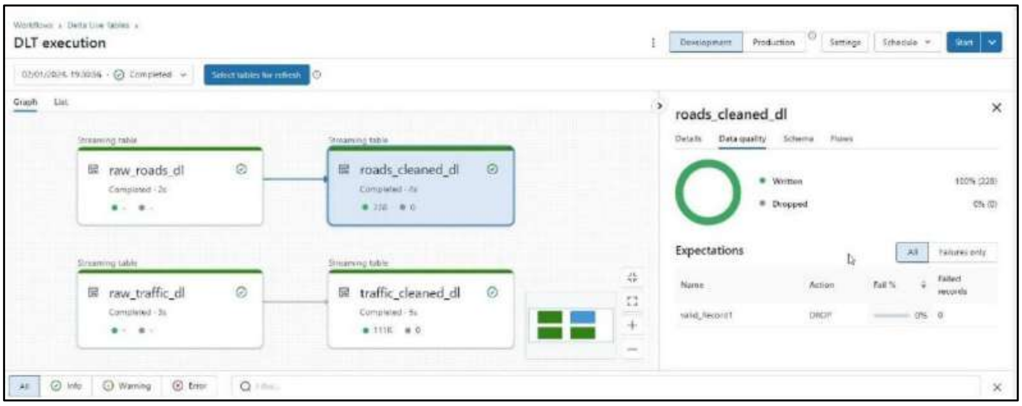


We can see 2 modes in DLT pipeline:

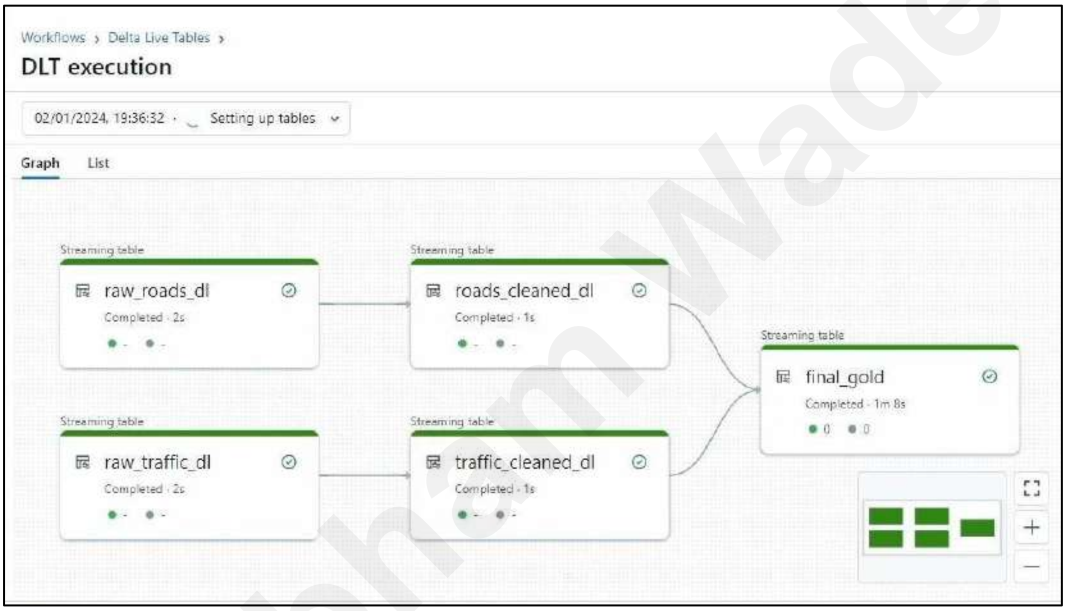
- **Development:** It retains the cluster for 2 hours. It does not make any retry.
- **Production:** It invokes the cluster, and it stops the cluster once the execution is completed. It does retry as well.
- Through this DLT pipeline we need only 1 table and we will join both the tables after passing data quality checks.
- These are certain steps taken by Delta Live Table.

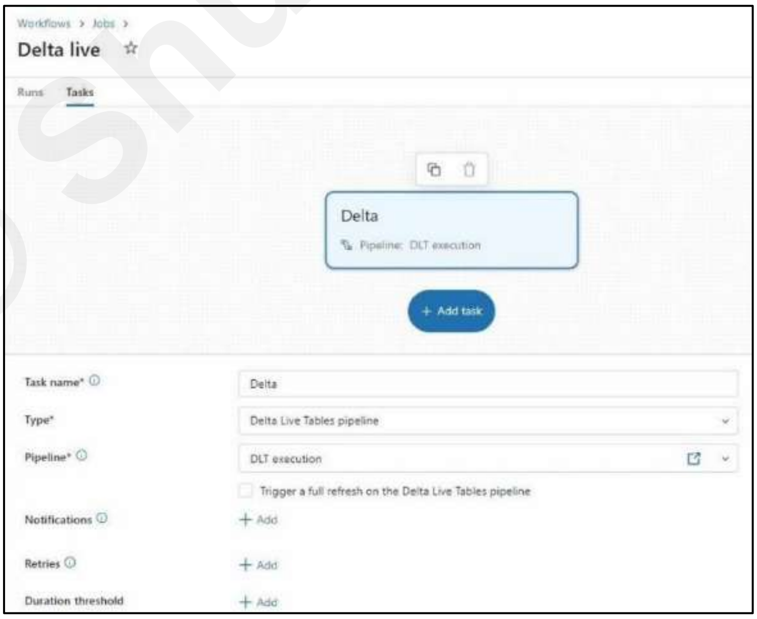
- Data quality metric is also visible in DLT. We can see the name of the constraint as valid, and it shows the percentage of failure.

© Shubham Wadekar



- In the final gold table, we need only a few selected columns, so we select them and join both the tables in DLT.





- We can create a job for DLT as well as shown below.

© Shubham Wadekar

Conclusion

This project successfully demonstrates the design and implementation of a modern data lakehouse architecture using Azure Databricks, Delta Lake, and Azure Data Lake Storage Gen2 for managing and analyzing traffic and road datasets.

By following the medallion architecture (Bronze → Silver → Gold), we ensured a structured and scalable data pipeline that supports both batch and real-time data ingestion using Spark Structured Streaming with Auto Loader.

Key achievements include:

- Secure and governed access using Access Connector and Unity Catalog.
- Cleaned and transformed data in the Silver Layer with derived metrics like Electric_Vehicles_Count and Vehicle_Intensity.
- Creation of business-ready Gold Layer tables optimized for analytics and reporting.
- Integration with Power BI to generate interactive dashboards that provide insights into traffic patterns, road usage, and electric vehicle trends.
- Implementation of CI/CD pipelines using Azure DevOps, enabling version control and automated deployments across environments.

This end-to-end pipeline not only enables real-time analytics but also serves as a reusable framework for building similar data-driven solutions in the transportation or smart city domain.

© Shubham Wadekar