Delta Live Tables (DLT) in Databricks: A Comprehensive Guide

Introduction

Delta Live Tables (DLT) is a framework in Databricks that simplifies ETL (Extract, Transform, Load) pipelines with declarative data transformations. It allows for data processing with built-in reliability, performance optimizations, and data quality enforcement.

Step 1: Setting Up Required Resources

To start with DLT, create the following resources:

1. **Resource Group:** Create a resource group in the Azure portal.
2. **Storage Account (Data Lake):** Ensure hierarchical namespace is selected.
3. **Databricks:** Create a Databricks workspace.
4. **Access Connector:** Create an access connector for secure connectivity between Databricks and the Data Lake.

Step 2: Assign Permissions

1. Navigate to the Data Lake storage account.
2. Go to **IAM (Identity and Access Management)** settings.
3. Assign **Storage Blob Contributor** role to the Access Connector to enable reading/writing data.

Step 3: Setting Up Unity Catalog

1. Open Databricks workspace.
2. Create a **Unity Metastore** to enable the **Unity Catalog**.
3. Ensure the metastore has a specified storage location.

Step 4: Configuring External Locations

Before creating a cluster, set up an **External Location**:

- This enables Databricks to read/write data to the Data Lake securely.
- Permissions should be scoped at the **container level**.

Steps to create an external location:

1. Assign credentials.
2. Create an external location using those credentials.

Step 5: Creating a Compute Cluster

Databricks manages virtual machines that act as **worker and driver nodes** for running computations.

Step 6: Creating a Databricks Workspace and Notebook

1. Open Databricks and create a **notebook**.

2. Connect the notebook to the cluster.
3. Create a Unity Catalog schema.

```
CREATE SCHEMA dtl_catalog.raw;
```

4. Create a Delta table (external source table):

```
CREATE TABLE dtl_catalog.raw.raw_customers (
  id INT,
  name STRING,
  salary INT,
  email STRING
) USING DELTA
LOCATION 'abfss://raw@datalake.dfs.core.windows.net/raw_customers';
```

5. Insert data into the table:

```
INSERT INTO dtl_catalog.raw.raw_customers
VALUES
(1, 'John', 50, 'john@gmail.com'),
(2, 'Jane', 60, 'jane@gmail.com');
```

6. Verify data storage in Delta format:

```
SELECT * FROM dtl_catalog.raw.raw_customers;
```

Delta Live Tables (DLT) Concepts

DLT enables building **incremental** and **batch pipelines**. It supports:

1. **Streaming Tables**: Process real-time data streams.
2. **Materialized Views**: Store precomputed results.
3. **Views**: Logical transformations without storing results.

Creating Delta Live Tables Pipeline

1. Create a **Bronze Layer** (Streaming Table):
2. Create a **Silver Layer** (View):
3. Create a **Gold Layer** (Materialized View):

Once configured, running the pipeline will automatically process the data.

Table Renaming & Handling Data Changes

1. Renaming a table in DLT creates a **new table** while marking the old one as **tombstoned**.
2. Use **table ID** and **Delta logs** to track changes.

3. Streaming tables should always read from an **append-only source**.

Append Flow Example

1. Suppose a new source table raw_customers_new is added.
2. Perform a **Union** operation between raw_customers and raw_customers_new.
3. Append the results to the **gold layer**.

Using Parameters in Delta Live Tables

1. Open **Delta Live Tables** settings.
2. Navigate to **Configuration** and add parameters.
3. Run the pipeline dynamically based on these parameters.

Apply Changes API for Change Data Capture (CDC)

DLT supports **SCD Type 1 and Type 2** transformations:

1. **SCD Type 1 (UPSERT):** Overwrites existing records.
2. **SCD Type 2:** Maintains historical records with start and end dates.

To apply changes, use the **Apply Changes API**.

Ensuring Data Quality with Expectations API

DLT provides **Expectations API** to validate data quality.

- **WARN**: Flags issues but continues processing.
- **DROP**: Removes invalid records.
- **FAIL**: Stops the pipeline on error.

Example of Expectations API:

```
@dlt.table
def customers():
  return (
    spark.readStream.format("delta").load("path/to/data")
    .expect("id IS NOT NULL", "fail")
  )
```

Using Auto Loader for Non-Delta Sources

1. **Auto Loader** helps process non-Delta sources (CSV, JSON, Parquet).
2. Instead of a **streaming Delta table**, create a **Volume** under Unity Catalog.
3. Upload CSV files into the **Volume**.
4. Use Auto Loader to read data dynamically.

Example:

```
from pyspark.sql.functions import *


df = spark.readStream.format("cloudFiles")\
  .option("cloudFiles.format", "csv")\
  .load("/Volumes/myvolume")
```

Conclusion

Delta Live Tables (DLT) simplifies data pipelines in Databricks by providing:

- Incremental and batch data processing.
- Reliable data transformations with auto-validation.
- Unified data management using **Unity Catalog**.
- Streaming capabilities with **Auto Loader**.

By leveraging DLT, organizations can build scalable, maintainable, and automated data processing workflows with high reliability.