**Delta Live Tables (DLT) - Advanced Lab Guide**

**Lab 1: Understanding DLT YAML Anatomy**

**Objective:**  
Understand how to define and configure a DLT pipeline using a YAML file.

**Steps:**

1. Go to your Databricks workspace.
2. Navigate to **Workflows > Delta Live Tables**.
3. Click **Create Pipeline** > **Create from YAML**.
4. Use the sample YAML:

yaml

name: sample\_pipeline

clusters:

- label: default

autoscale:

min\_workers: 1

max\_workers: 5

libraries:

- notebook: /Workspace/Shared/sample\_notebook

configuration:

param1: value1

1. Create a notebook (sample\_notebook) with basic DLT logic.
2. Deploy the pipeline and observe the DAG.

**Expected Outcome:**  
Pipeline runs as per YAML definition; shows tables defined in the notebook.

**Lab 2: Using Auto Loader in DLT**

**Objective:**  
Ingest streaming data using Auto Loader in a DLT pipeline.

**Steps:**

1. Create a notebook with this code:

python

df = (spark.readStream

.format("cloudFiles")

.option("cloudFiles.format", "csv")

.load("/mnt/bronze/input/"))

@dlt.table

def raw\_csv():

return df

1. Configure the pipeline to include this notebook.
2. Drop some sample CSV files into /mnt/bronze/input/.

**Expected Outcome:**  
New files are automatically ingested into raw\_csv DLT table.

**Lab 3: Applying EXPECT Rules in DLT**

**Objective:**  
Use EXPECT clauses to apply data quality rules.

**Steps:**

1. Add the following to a DLT table:

sql

CREATE OR REFRESH LIVE clean\_data AS

SELECT \* FROM LIVE.raw\_csv

EXPECT colA IS NOT NULL ON VIOLATION DROP ROW

1. Add input data with valid and invalid values for colA.

**Expected Outcome:**  
Rows where colA IS NULL are dropped; pipeline logs this event.

**Extensions:**

* Use ON VIOLATION FAIL UPDATE or QUARANTINE INTO.

**Lab 4: Ingesting Change Data Capture (CDC)**

**Objective:**  
Use APPLY CHANGES INTO for ingesting CDC data into target tables.

**Steps:**

1. Simulate CDC input data with INSERT, UPDATE, DELETE.
2. Define CDC logic:

sql

APPLY CHANGES INTO live.silver\_data

FROM stream.live.cdc\_source

KEYS (id)

SEQUENCE BY updated\_at

COLUMNS \* STORED AS SCD TYPE 1

1. Use streaming table cdc\_source to simulate changes.

**Expected Outcome:**  
Target table always reflects latest state of each business key.

**Lab 5: Monitoring Pipelines & Event Hooks**

**Objective:**  
Track pipeline health and respond to events (failures, warnings).

**Steps:**

1. Run the DLT pipeline and navigate to its **Monitoring UI**.
2. Use SQL to analyze logs:

sql

SELECT \* FROM dlt.event\_log WHERE event\_type = 'flow\_progress'

1. (Optional) Configure a webhook or Databricks Job triggered by pipeline failure.

**Expected Outcome:**  
You can audit metadata and get alerted on failures or schema issues.

**Lab 6: Schema Evolution with Auto Loader**

**Objective:**  
Demonstrate how Auto Loader handles schema changes.

**Steps:**

1. Initially ingest CSV files with 3 columns.
2. Introduce a new file with an additional 4th column.
3. In the notebook, configure:

python

.option("cloudFiles.schemaEvolutionMode", "rescue")

1. Optionally, turn on **Schema Inference** in pipeline settings.

**Expected Outcome:**  
New column is captured in the downstream DLT table schema automatically.

**Lab 7: Promote Pipelines Across Environments**

**Objective:**  
Deploy the same DLT pipeline in dev, staging, and production using YAML + CLI.

**Steps:**

1. Create multiple YAML files:
   * dlt\_dev.yaml
   * dlt\_prod.yaml
2. Use dbx or databricks CLI:

bash

dbx deploy --environment=prod

1. Store secrets and configs (e.g., source paths) in environment-specific key/value pairs.

**Expected Outcome:**  
Same pipeline logic gets promoted across environments with minimal changes.