**Delta Live Tables (DLT) – Advanced Study Guide**

**1. DLT YAML Anatomy**

**Purpose:**  
The YAML config file defines the structure and behavior of a DLT pipeline, enabling easy automation and promotion across environments.

**Key Sections:**

* **name**: Pipeline identifier
* **clusters**: Worker definitions for auto-scaling
* **libraries**: Notebooks or JARs used for defining tables
* **configuration**: Custom key-value pairs (e.g., paths, params)
* **channel**: (optional) Preview or current

**Example:**

yaml

name: bronze\_to\_silver\_pipeline

clusters:

- label: default

autoscale:

min\_workers: 2

max\_workers: 8

libraries:

- notebook: /Repos/user@org.com/bronze\_to\_silver

configuration:

source\_path: /mnt/raw/

env: dev

**2. Auto Loader in DLT**

**Purpose:**  
Efficient ingestion of cloud files (CSV, JSON, Avro, etc.) without needing manual schema inference or job orchestration.

**Key Features:**

* Incremental file detection using **directory listing** or **file notification**.
* Supports schema evolution and rescue mode.
* Works with S3, ADLS, GCS.

**Basic Usage:**

python

spark.readStream.format("cloudFiles") \

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

.load("/mnt/input/")

**DLT Example:**

python

@dlt.table

def bronze\_stream():

return (

spark.readStream.format("cloudFiles")

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

.load("/mnt/bronze")

)

**3. EXPECT Rules (fail/drop/quarantine)**

**Purpose:**  
Data quality checks embedded into the pipeline to enforce clean and trusted datasets.

**Options:**

* **FAIL UPDATE**: Stops the job if condition is violated.
* **DROP ROW**: Removes rows that don’t meet the condition.
* **QUARANTINE INTO**: Moves bad data into a quarantine table.

**Example:**

sql

CREATE OR REFRESH LIVE clean\_table

AS SELECT \* FROM LIVE.raw\_data

EXPECT id IS NOT NULL ON VIOLATION DROP ROW;

**Quarantine Example:**

sql

EXPECT col > 0 ON VIOLATION QUARANTINE INTO live.bad\_records;

**4. CDC Ingest Pattern**

**Purpose:**  
Efficiently process change data capture (insert/update/delete) from upstream sources using APPLY CHANGES INTO.

**Best For:**

* SCD Type 1/2
* Merge-on-read workloads
* Tracking data versioning from CDC logs (Debezium, SQL CDC, etc.)

**Syntax:**

sql

APPLY CHANGES INTO live.target\_table

FROM stream.live.source\_table

KEYS (business\_key)

SEQUENCE BY last\_updated

COLUMNS \* EXCEPT(op) STORED AS SCD TYPE 1;

**Notes:**

* SEQUENCE BY must have a monotonically increasing field like timestamp or offset.
* Works only with streaming sources.

**5. Pipeline Monitoring & Event Hooks**

**Monitoring:**

* Use **DLT UI** to inspect DAG, lineage, and metrics.
* Query event\_log for operational metadata.

sql

SELECT \* FROM dlt.event\_log

WHERE event\_type = 'flow\_progress'

**Event Hooks:**

* Trigger Jobs/alerts/webhooks on failure or success using Databricks Jobs or REST API.
* event\_log captures errors, schema changes, row counts, etc.

**Use Cases:**

* Notify via Slack/Webhook on failure
* Send metrics to observability tools (e.g., Datadog, Splunk)

**6. Schema Evolution**

**Problem:**  
Input data schema changes over time (e.g., new column added in source CSV).

**Auto Loader Solution:**

* Set .option("cloudFiles.schemaEvolutionMode", "rescue")
* Optionally allow schema inference:

python

spark.conf.set("spark.databricks.cloudFiles.schemaInference.enabled", "true")

**DLT Handling:**  
DLT will automatically evolve schema if enabled. New fields will appear in the downstream tables unless explicitly excluded.

**7. Promote Across Environments (Dev → QA → Prod)**

**Approach:**

* Use YAML + parameterized configs for pipeline portability.
* Maintain separate configs for each env:
  + dlt\_dev.yaml
  + dlt\_prod.yaml

**Techniques:**

* Use dbx or databricks-cli for promotion automation.
* Store configs in workspace secrets or files.

**Best Practices:**

* Avoid hardcoded paths.
* Externalize environment-specific configs.
* Use feature flags (env = dev) inside notebooks or SQL.

**Promotion Tools:**

* dbx deploy --environment=prod
* Azure DevOps/GitHub Actions + Databricks CLI