**Case Study: Deploying an Idempotent Streaming Job with Checkpointing and Trigger-Once Replay**

**Overview**

This case study outlines the end-to-end deployment of an **idempotent Spark Structured Streaming job** that utilizes **Delta Lake**, **checkpointing**, and **Trigger.Once** to ensure **exactly-once semantics**, **replay safety**, and **incremental recovery** in real-time data processing.

**Business Scenario**

An e-commerce company needs to build a daily data ingestion pipeline that:

* Ingests customer clickstream data from cloud storage (e.g., S3, Azure Blob, GCS)
* Writes to a Delta Lake gold layer table
* Guarantees idempotent data loads with no duplication
* Supports recovery and reprocessing from intermediate states using checkpointing
* Executes once a day on new data

**Objectives**

* Deploy a **replayable** batch-style stream processing job using **Trigger.Once**
* Ensure **idempotency** with Delta ACID semantics
* Leverage **checkpointing** for recovery and lineage
* Integrate with **Auto Loader** for scalable file ingestion

**Architecture**

**Components:**

* **Spark Structured Streaming** (Databricks / EMR / Synapse)
* **Delta Lake** for sink
* **Auto Loader** for input
* **Trigger.Once** for deterministic execution
* **Checkpoint Directory** in cloud storage

+----------------------+ +--------------------+ +--------------------+

| Cloud Storage (S3) | ---> | Auto Loader Stream | ---> | Delta Gold Table |

+----------------------+ +--------------------+ +--------------------+

| ↑

↓ |

Checkpoint Directory ---------+

**Step-by-Step Implementation**

**Step 1: Define Input Path and Storage Location**

val inputPath = "/mnt/raw/clickstream/"

val outputPath = "/mnt/delta/gold/clickstream/"

val chkptPath = "/mnt/chkpt/clickstream\_once/"

**Step 2: Read Streaming Data with Auto Loader**

val clickDF = spark.readStream

.format("cloudFiles")

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

.option("cloudFiles.schemaLocation", "/mnt/schema/clickstream/")

.load(inputPath)

**Step 3: Data Transformations**

val cleanDF = clickDF.selectExpr("userId", "eventType", "timestamp")

.withColumn("eventTime", to\_timestamp(col("timestamp")))

**Step 4: Write to Delta with Checkpointing and Trigger.Once**

cleanDF.writeStream

.format("delta")

.outputMode("append")

.option("checkpointLocation", chkptPath)

.trigger(Trigger.Once())

.start(outputPath)

.awaitTermination()

**How Idempotency is Ensured**

1. **Delta Format:** Guarantees ACID transactions.
2. **Trigger.Once:** Runs a deterministic batch; prevents multiple mini-batches.
3. **Checkpointing:** Tracks processed files and offsets for Auto Loader.
4. **Append Mode:** Avoids overwrites.

**Recovery Scenario**

In case of failure:

* Re-run the same job with the same checkpoint and schema path.
* Auto Loader will resume file discovery from last offset.
* Delta ensures no duplicate writes due to ACID.

**Validation of Exactly-Once Behavior**

1. **Trigger Job Twice**: Using same checkpoint path
2. **Check Delta Table History**:

DESCRIBE HISTORY delta.`/mnt/delta/gold/clickstream/`

1. **Validate Record Count Stability**

**Benefits Realized**

* **No duplicate ingestion** even on job restarts
* **Resumable**: Failures can be recovered from last good state
* **Scalable ingestion** using Auto Loader
* **Daily replay** using Trigger.Once() enables simplified orchestration

**Conclusion**

This deployment pattern—combining **Auto Loader**, **Delta Lake**, **Trigger.Once**, and **checkpointing**—ensures a **resilient**, **replay-safe**, and **idempotent** batch-style streaming pipeline suitable for scheduled ingestion and backfill workflows.

**Extensions**

* Add schema evolution support with mergeSchema
* Integrate with job scheduler (e.g., Azure Data Factory, Airflow, DB workflows)
* Use MERGE INTO for deduplication if needed

**References**

* <https://docs.delta.io/latest/delta-streaming.html>
* <https://learn.microsoft.com/en-us/azure/databricks/ingestion/auto-loader>
* <https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html>