## ****Ingestion Guardian****

### Objective

Automate data quality checks, pipeline validations, and rollback strategies in a Databricks-based data pipeline to prevent human error, ensure reliability, and enable consistent recovery without relying on memory or manual steps.

## 1 Overview

### Use Case

Daily ingestion of payroll timesheet files into a Delta Lake table. These files must be validated for:

* Presence and completeness (non-empty)
* Schema integrity
* Duplicate prevention (idempotency)
* Pipeline recovery if something goes wrong
* Invalid records routing to DLQ

## 2 Architecture Components

* **Bronze Zone:** Raw file ingestion
* **Silver Zone:** Validated and deduplicated records
* **DLQ:** Dead-letter queue for invalid or corrupted records
* **Checkpoints:** For retry-safe streaming
* **Slack/Email Alerts:** For anomalies (empty files, schema mismatch, DLQ growth)

## 3 Automation Components & Code

### A. File Ingestion + Schema Validation

from pyspark.sql.functions import input\_file\_name

from pyspark.sql.utils import AnalysisException

from datetime import date

file\_date = date.today().isoformat()

input\_path = f"/mnt/raw/payroll/payroll\_{file\_date}.csv"

try:

df = spark.read.option("header", "true").csv(input\_path)

df = df.withColumn("source\_file", input\_file\_name())

except AnalysisException:

raise Exception(f"File not found: {input\_path}")

# Circuit Breaker

expected\_cols = {"employee\_id", "salary", "department", "timestamp"}

if df.count() == 0 or set(df.columns) != expected\_cols:

raise Exception("Circuit Breaker: Empty or schema mismatch")

### B. Data Validation & Dead-Letter Queue (DLQ)

valid\_df = df.filter("employee\_id IS NOT NULL AND salary IS NOT NULL")

invalid\_df = df.subtract(valid\_df)

# Save invalid records to DLQ

invalid\_df.write.mode("append").parquet("/mnt/dlq/payroll/")

### C. Idempotent Upsert with Delta Lake

from delta.tables import DeltaTable

from pyspark.sql.functions import md5, concat\_ws

valid\_df = valid\_df.withColumn("checksum", md5(concat\_ws("||", \*valid\_df.columns)))

target\_table = DeltaTable.forPath(spark, "/mnt/delta/payroll")

target\_table.alias("target").merge(

valid\_df.alias("source"),

"target.employee\_id = source.employee\_id"

).whenMatchedUpdateAll().whenNotMatchedInsertAll().execute()

### D. Retry-safe Streaming Pipeline

stream\_df = spark.readStream.option("header", "true").schema(valid\_df.schema).csv("/mnt/raw/payroll/")

stream\_df.writeStream \

.format("delta") \

.outputMode("append") \

.option("checkpointLocation", "/mnt/checkpoints/payroll/") \

.start("/mnt/delta/payroll/")

## 4 Rollback Strategy (on corruption or bad load)

-- Restore to a previous version

RESTORE TABLE payroll TO VERSION AS OF 158;

-- Or TIME based

RESTORE TABLE payroll TO TIMESTAMP AS OF '2025-07-10 08:00:00';

## 5 Monitoring & Alerting

* Use **Unity Catalog audit logs** or **Auto Loader metrics** to monitor file arrival.
* Trigger Slack/email alerts using Databricks Jobs UI or webhooks when:
  + Empty files detected
  + Schema mismatch
  + DLQ grows beyond threshold

## 6 Data Contracts Are Gold

**Objective**

Define clear expectations between data producers and consumers by formalizing schema, SLA, and ownership.

**Data Contract Definition Table**

|  |  |  |
| --- | --- | --- |
| Attribute | Description | Example / Value |
| Dataset Name | Logical name of the dataset | payroll\_timesheet\_daily |
| Producer Team | Team/system responsible for producing the data | HR Tech Team |
| Consumer Team | Team/system consuming the data | Payroll Processing Team |
| Schema Definition | Formal schema with data types and constraints | See schema contract (e.g., employee\_id STRING NOT NULL) |
| Schema Versioning | Version control process and location | Git repo: contracts/payroll\_timesheet\_v2.json |
| Delivery SLA | Agreed delivery schedule or window | Daily by 8 AM IST |
| Data Freshness SLA | Max acceptable delay between event and availability | ≤ 1 hour from event time |
| Contact (Producer) | Name/email/Slack of producer POC | hr-data@company.com |
| Contact (Consumer) | Name/email/Slack of consumer POC | payroll-support@company.com |
| Breaking Change Policy | Agreement on how schema changes are communicated and validated | 2-week notice + signed-off schema update |
| DLQ Handling Agreement | Who is responsible for invalid data records | HR Tech Team will fix and reprocess |
| Change Notification Channel | Slack/email group where all changes are posted | #payroll-ingestion-alerts |

**Key Practices**

* **Schema Contracts**: Defined in .avro or .json and version-controlled in Git.
* **Column-level Requirements**: Mark nullable vs required fields (e.g., employee\_id NOT NULL).
* **SLA Definitions**: Document expectations (e.g., file delivery by 8 AM IST daily).
* **Ownership**: Assign teams for both data production (HR System) and consumption (Payroll Engine).
* **Breaking Change Policy**: No schema updates without prior communication and agreement.

**Communication Framework**

* Publish contracts in a shared Confluence page or schema registry.
* Enable Slack channels or MS Teams integration for alerts, questions, and change notifications.

**Risk Mitigation**

* Prevent downstream crashes from unexpected schema changes.
* Ensure that consumers know exactly what data to expect and when.
* Reduce rework and escalations from unclear assumptions.

**Sleep Isn’t Lazy — It’s Strategic**

**Why It Matters**

Burnout kills creativity and judgment. Sleep-deprived engineers ship fragile systems. A well-rested team builds reliable, testable, and support-free pipelines.

**Design Mindset**

Ask yourself daily: **“Am I building a pipeline that survives the night?”**

**Best Practices**

* Build with observability: logs, metrics, and alerts.
* Design idempotent steps so retries are safe.
* Include circuit breakers and dead-letter queues to isolate errors.
* Use automatic rollback features to restore state.

**Team Culture Tip**

Make rest part of the engineering process. Prioritize automation not just for scale, but for **peace of mind**.

The best engineers don’t wake up at 3 AM. Their systems wake up for them — and heal themselves.

## Final Notes

Automation isn't luxury—it's survival. Human memory fails, but code doesn’t forget. This pattern ensures reliable, repeatable, and recoverable data ingestion without manual intervention.

Data contracts are the backbone of trust. Define them clearly, honor them consistently, and communicate them openly.

Rested teams build resilient systems. Prioritize maintainability over heroics.