Databricks: Payroll Shield

Payroll Pipeline Observaility & Reliability

Title: **Payroll Shield: Reliable and Observable Ingestion for Timesheet Data**

Type: **Data Quality Enforcement with Observability**

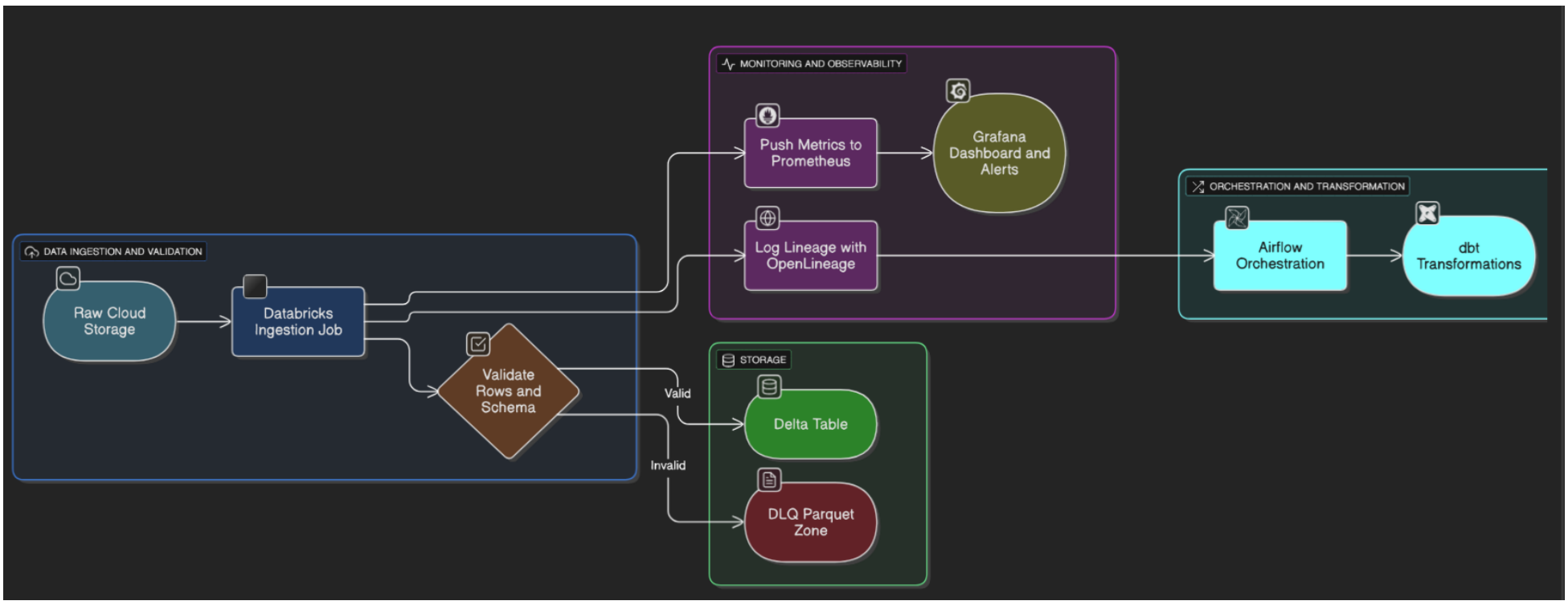
Use Case: Prevent payroll errors by detecting and alerting on empty or invalid timesheet files during ingestion, while ensuring idempotent, retry-safe processing with full observability (logs, metrics, lineage, and alerts).

Business Executive Summary:

Payroll systems rely on accurate and timely data ingestion from daily timesheet files. Empty or corrupt files can lead to missed payments, audit failures, and employee dissatisfaction. This solution ensures robust ingestion through idempotent processing, circuit breakers, failover paths, and dead-letter queues, integrated with observability using Airflow, dbt, Prometheus, Grafana, and OpenLineage.

By building a culture of visibility, this design enables early detection, real-time alerting, and complete traceability, reducing operational risk and increasing confidence in payroll accuracy.

Architecture Overview



Step-by-Step Implementation

1. Ingest File with Circuit Breaker and Failover

```
from pyspark.sql.utils import AnalysisException
from datetime import date

file_date = date.today().isoformat()
primary_path = f"/mnt/raw/payroll/payroll_{file_date}.csv"
fallback_path = "/mnt/raw/payroll/fallback/latest_backup.csv"

try:
    df = spark.read.option("header", "true").csv(primary_path)
    print("Primary file loaded.")
except AnalysisException:
    df = spark.read.option("header", "true").csv(fallback_path)
    print("Fallback file loaded.")
```

2. Circuit Breaker Validation

```
expected_columns = {"employee_id", "salary", "department", "timestamp"}
if df.count() == 0 or set(df.columns) != expected_columns:
    raise Exception("Circuit Breaker: Empty file or schema mismatch.")
```

3. Data Quality Filter (Valid vs Invalid)

```
from pyspark.sql.functions import input_file_name

df = df.withColumn("source_file", input_file_name())

valid_df = df.filter("employee_id IS NOT NULL AND salary IS NOT NULL")
invalid_df = df.subtract(valid_df)

invalid_df.write.mode("append").parquet("/mnt/dlq/payroll/")
```

Databricks: Payroll Shield

4. Idempotent UPSERT into Delta Lake

```
from pyspark.sql.functions import md5, concat_ws
from delta.tables import DeltaTable

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()
```

5. Emit Prometheus Metrics

```
from prometheus_client import CollectorRegistry, Gauge, push_to_gateway

registry = CollectorRegistry()
row_count_metric = Gauge('payroll_valid_rows', 'Number of valid payroll rows', registry=registry)
row_count_metric.set(valid_df.count())

push_to_gateway('http://<prometheus-host>:9091', job='payroll_ingestion', registry=registry)
```

6. Slack Alert on DLQ

```
import requests

def send_slack_alert(message):
    webhook_url = "<your_slack_webhook_url>"
    payload = {"text": message}
    requests.post(webhook_url, json=payload)

if invalid_df.count() > 0:
    send_slack_alert(f"Payroll DLQ Alert: {invalid_df.count()} rows in DLQ on {file_date}")
```

7. Orchestrate with Airflow

```
from airflow import DAG
from airflow.providers.databricks.operators.databricks import DatabricksRunNowOperator
from datetime import datetime

with DAG("payroll_pipeline_dag", start_date=datetime(2024, 1, 1), schedule_interval="0 8 * * *", catchup=False) as dag:
    run_pipeline = DatabricksRunNowOperator(
        task_id="run_payroll_job",
        databricks_conn_id="databricks_default",
        job_id=12345
    )
```

8. dbt Models for Transformations

Model: stg_payroll_validated.sql

```
SELECT *
FROM {{ source('delta_lake', 'payroll') }}
WHERE salary IS NOT NULL AND employee_id IS NOT NULL
```

Model: final_payroll_agg.sql

```
SELECT department, COUNT(*) as employee_count, AVG(salary) as avg_salary
FROM {{ ref('stg_payroll_validated') }}
GROUP BY department
```

9. Lineage Tracking with OpenLineage + dbt

OpenLineage Config (profiles.yml)

```
openlineage:
  transport:
    type: http
    url: http://<openlineage-host>:5000
```

Run with Lineage:

```
OPENLINEAGE_DISABLED=false dbt run
```

Grafana Dashboard Panels

- Valid payroll row count
- DLQ record volume
- Ingestion runtime
- Files processed per day
- Error trends over time

Databricks: Payroll Shield

Final Outcome

Component	Description
Circuit Breaker	Stops ingestion on empty or invalid files
Retry Safe	Delta Lake with checkpointing and idempotency
DLQ Handling	Bad records safely logged and stored
Alerting	Slack notification on DLQ detection
Metrics	Row counts, latency pushed to Prometheus
Visualization	Grafana panels for all quality signals
Lineage	OpenLineage metadata via dbt & Airflow
Orchestration	Airflow DAG for end-to-end pipeline