Resilience over Perfection

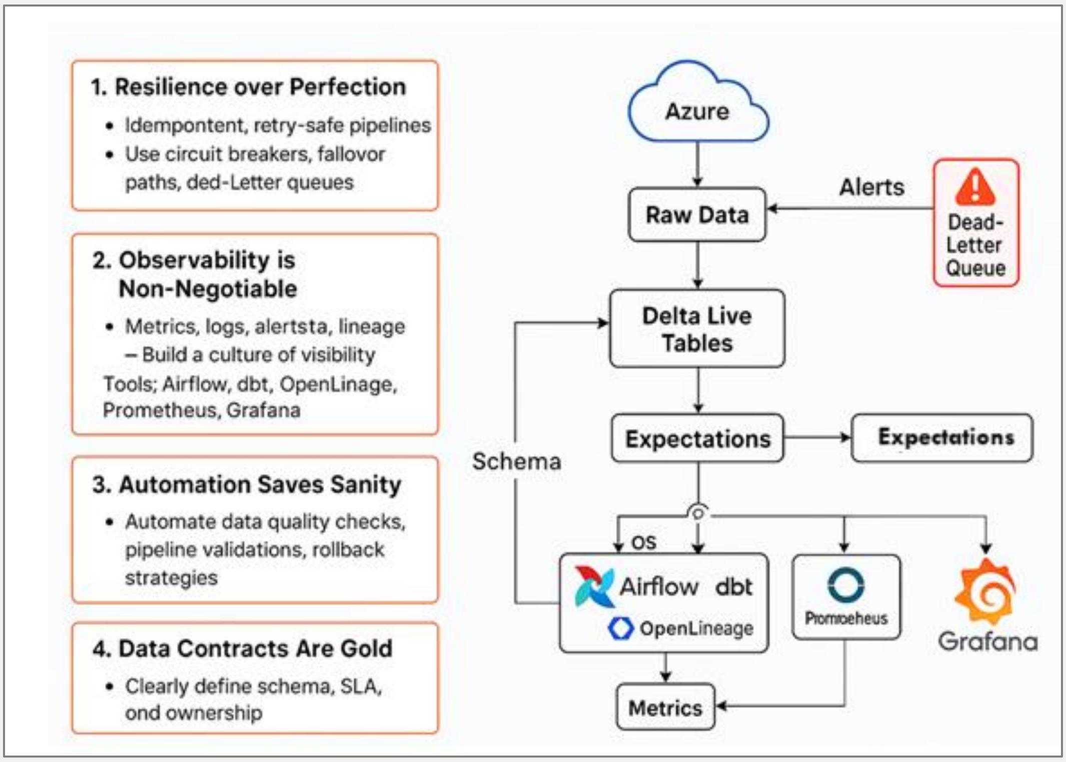
This Databricks-based Zomato-like ordering platform is built for real-time, reliable, and intelligent food delivery operations. It ensures **resilient pipelines** with idempotency, dead-letter queues, and circuit breakers, while offering **deep observability** via logs, metrics, and lineage tools like Airflow and Prometheus. Automated data quality checks and enforced data contracts prevent downstream failures. The system supports ML-powered personalization and delivery insights, enabling fast, scalable decision-making.

Type: **Real-time Data Platform** using **Lakehouse Architecture** on Databricks.

Use Case: Powering a Zomato-like food ordering ecosystem with real-time order processing, delivery tracking, restaurant analytics, customer reviews, and ML-driven dish recommendations — all with resilient pipelines, automated validation, and enterprise-grade observability.

Design Goals: Idempotent logic, retry-safe architecture, circuit breakers, and dead-letter queues.

Architecture:



Delta Live Tables (DLT) with Idempotency & Dead-Letter Queue

python

CopyEdit

dlt_order_pipeline.py

import dlt

from pyspark.sql.functions import col, input_file_name, current_timestamp, lit

1. Raw ingestion from Autoloader

```
@dlt.table(name="orders_raw", comment="Raw food orders")
def ingest_orders():
    return (
        spark.readStream.format("cloudFiles")
        .option("cloudFiles.format", "json")
        .load("/mnt/zomato-data/raw/orders/")
        .withColumn("source_file", input_file_name())
        .withColumn("ingested_at", current_timestamp())
    )
```

 # 2. Apply data quality checks (circuit breaker + DLQ)

```
@dlt.table(name="orders_valid", comment="Validated food orders")
@dlt.expect("has_user_id", "user_id IS NOT NULL")
@dlt.expect("has_dish_id", "dish_id IS NOT NULL")
def validate_orders():
    return dlt.read_stream("orders_raw").dropDuplicates(["order_id"])

@dlt.table(name="orders_dlq", comment="Failed records")
def failed_orders():
    return dlt.read_stream("orders_raw").filter(
        col("user_id").isNull() | col("dish_id").isNull()
    )
```

Retry-safe Jobs & Circuit Breaker Using Databricks Workflows

python

CopyEdit

```
# circuit_breaker.py

from pyspark.sql.functions import count
from pyspark.sql.utils import AnalysisException
try:
    df = spark.read.format("delta").table("orders_valid")
    record_count = df.count()
```

```
    if record_count == 0:
        raise Exception("🚨 Circuit Breaker Triggered: Zero valid records.")
except AnalysisException as e:
    raise Exception("Delta Table Not Found. Retry safe halt.")
```

Use this as a pre-task notebook in Workflows → if circuit trips, halt downstream tasks.

---

2. Observability is Non-Negotiable

Design Goals: Logs, metrics, lineage. Enable complete visibility.

Logging to Delta Table

python

CopyEdit

```
# logging.py
from pyspark.sql import Row
import datetime
def log_event(component, status, message):
    log_df = spark.createDataFrame(
        [Row(timestamp=datetime.datetime.utcnow(), component=component, status=status,
message=message)]
    )
    log_df.write.mode("append").format("delta").saveAsTable("platform_logs")
# Example Usage
log_event("orders_valid", "SUCCESS", "2000 records processed.")
```

Airflow + OpenLineage + Prometheus/Grafana

- Integrate Airflow DAGs with OpenLineage emitters.
- Emit metrics from Spark via custom Prometheus endpoints.
- Visualize pipeline duration, file lag, and failure rates in Grafana.

---

3. Automation Saves Sanity

Design Goals: Auto checks, auto rollback, and validation

Data Quality Automation with Expectations

Already shown above in DLT (@dlt.expect(...)). Add more:

python

CopyEdit

```
@dlt.expect_or_drop("valid_price", "price > 0")
@dlt.expect_or_drop("valid_quantity", "quantity > 0")
```

Auto-Rollback Example on Failure

python

CopyEdit

```python
# rollback_if_failure.py

try:
    df = spark.read.table("orders_valid")
    df.write.mode("overwrite").saveAsTable("orders_snapshot")
    log_event("snapshot", "SUCCESS", "Backup created.")
except Exception as e:
    log_event("snapshot", "FAILURE", str(e))
    spark.sql("RESTORE TABLE orders_valid TO VERSION AS OF (current_timestamp() - INTERVAL 1 HOUR)")
```

---

4. Data Contracts Are Gold

Design Goals: Schema ownership, enforced contracts, SLAs

Enforce Schema with Autoloader

python

CopyEdit

```python
from pyspark.sql.types import StructType, StringType, IntegerType, TimestampType

order_schema = StructType() \
    .add("order_id", StringType()) \
    .add("user_id", StringType()) \
    .add("dish_id", StringType()) \
    .add("price", IntegerType()) \
    .add("order_time", TimestampType())

df = (
    spark.readStream.format("cloudFiles")
    .option("cloudFiles.format", "json")
    .schema(order_schema)
```

```
    .load("/mnt/zomato-data/raw/orders/")
)
```

Data SLA Monitoring

python

CopyEdit

```
# sla_monitor.py

from pyspark.sql.functions import max, current_timestamp

df = spark.read.table("orders_valid")
max_time = df.select(max("order_time")).collect()[0][0]

delay_minutes = (datetime.datetime.utcnow() - max_time).total_seconds() / 60
if delay_minutes > 10:
    log_event("SLA_MONITOR", "ALERT", f"Last data point was {delay_minutes} mins ago.")
```

Optional: DAG in Airflow (External Scheduling)

python

CopyEdit

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

with DAG("zomato_pipeline", start_date=datetime(2025, 7, 1), schedule_interval="5 * * * *") as dag:
    run_job = DatabricksRunNowOperator(
        task_id="run_zomato_dlt_job",
        databricks_conn_id="databricks_default",
        job_id=1234  # Replace with your actual job ID
    )
```

Summary Table

Databricks: FoodOps Lakehouse

Principle	Feature Implemented
Resilience over Perfection	Idempotent DLT, DLQ, circuit breakers
Observability	Logs, alerts, OpenLineage + Prometheus
Automation Saves Sanity	Auto rollback, data quality validation
Data Contracts Are Gold	Schema enforcement, SLA monitoring

Final Note: This Zomato-style ordering platform on Databricks delivers a resilient, automated, and observable data foundation that supports real-time decision-making, customer personalization, and operational excellence. Built with strong data contracts and modern engineering practices, it's designed to scale reliably as business demand grows.