TMT Data Quality Framework Design

By Muchamad Fajar Alif

Summary

The framework outlines the comprehensive data quality platform architecture for TMT's ride-hailing data infrastructure, processing 200K+ daily transactions. The solution integrates three core features: Data Observability, Data Monitoring, and Data Alerting.

Architecture

MySQL (Source) → **Airflow** (Orchestration) → **GCS** (Staging) → **BigQuery** (DWH) → **Tableau** (Visualization) → **Data Quality Engine** (GCP Services) → **Alert System** (Slack/Telegram)

Core Framework Components

- Configuration Management System
- Results Storage Layer
- Alert Orchestration
- Tableau Data Mart Integration

Key Features

Data Observability

Monitor completeness and timeliness (freshness) automatically.

Core Observability Components

1. Freshness Monitoring Engine

Purpose: Track data arrival times and processing delays across all pipeline stages.

Idea: Each table that will be ingested to BigQuery stored in GCS first (Parquet), for the freshness we can check on the Parquet files modification time (landing time). For that we need one table as reference that stores information like table name, file name (prefix with date or batch identifier), and its SLA like 6 AM. From that we can have one service that will check if the table arrived according to its SLA.

For the processing delays, here we can add some buffer, if the table usually done ingested to GCS then to BigQuery at 6 AM, to accommodate late incoming data for example because of some upstream issue we can set the freshness monitor to check at the SLA+buffer time if the mentioned files still meet the time agreement.

Unique components:

- **Reference Table (SLA Configuration)**
  - Stores table name, GCS paths, SLA times (6 AM), buffers (30 min)
  - Configurable per table: transactions ready by 6:00 AM + 30min buffer
- **File Discovery Service**
  - Scans GCS for expected Parquet files
  - Extracts modification time (landing time) from file metadata
  - Handles file naming patterns with date/batch identifiers
- **SLA Evaluation Engine**
  - Compares file modification time vs SLA + buffer
  - Key logic: Check at SLA+buffer time if files meet agreement (can be hardcoded result (OK, Not OK) or some threshold calculation
  - Final Status: SLA_MET, PENDING, SLA_VIOLATED, CRITICAL_MISSING
- **Results Storage Table**
  - Stores all SLA check results with timestamps
  - Partitioned by date, clustered by table/status for performance

Freshness Monitoring Airflow DAG:

- **DAG design**: start → loops through SLA reference table → discovers files → evaluates SLA → stores results → sends alert → end
- **Schedule**: Configurable (short interval by default)

2. Completeness Scoring System

Purpose: Monitor data completeness ratios and identify missing data patterns.

Idea: For this, we can use configurable completeness rules that we can check for each table, for example transactions table would need to check NULL values for key columns, and users table would need to check different criteria at different columns as the completeness metrics.

The idea here is we can have some configs for each table in YAML that can define each completeness metrics whether it will be using some query builder template or raw sql query, the key is pairing each table with its specific configuration like this:

```
completeness_metrics:
  transaction_id_null_check:
    query_template: |
      SELECT COUNT(*) as null_count
      FROM `{dataset}.{table_name}`
      WHERE business_date = CURRENT_DATE('Asia/Jakarta')
        AND transaction_id IS NULL
    threshold:
      max_allowed: 0
      severity: "critical"
```

Airflow Integration

- Completeness can be scheduled after freshness monitoring for the related tables is done (dependency)
- Send alerts for failures or stop from sending the data to downstream
- **DAG design**: check_today_freshness (sensor) → start → fetch config → parse config → evaluates completeness → stores results → sends alerts → end
- **Schedule**: Configurable

3. Volume Anomaly Detection

Purpose: Detect unusual data volume patterns that indicate pipeline issues.

Idea: For the volume anomaly we can check it by using its trend and some threshold with the table data history itself as a baseline, as for the baseline, we can get it by profiling the table historically, and set good and bad thresholds from that. Use 3-6 months of historical data to establish realistic volume baselines, then detect anomalies when current volumes deviate significantly from normal patterns.

Unique Components:

- **Historical Data**
  - Needed to track volume or value trends to be compared with recent or latest data
- **Baseline Calculator Service**
  - Needed to have a standardized query to check repetitive columns that can be reused on other tables.

Airflow Integration

- **DAG design**: check_today_freshness → start → fetch config → parse config → evaluates volume anomaly → stores results → sends alerts → end
- **Schedule**: Configurable

4. Schema Evolution Tracker

Purpose: Monitor structural changes in data that might affect downstream processes.

Idea: We need to check schema changes early, so I think it's suitable to check or track schema on GCS and BigQuery, we can have these type of changes related to schema (new columns, deleted columns, and data type changes) and these three may can have different severity based on the tables and need different action some times (allow and notify, block and alert).

Key components for this:

- **Configs repository**
  - Stores all configs and can be synced to the Airflow environment
  - Versioning so even business user can track their configs
- **Configuration parser**
  - Load YAML file to get metrics, thresholds, and query that will be executed on BigQuery then evaluate result with the threshold
- **Airflow Integration**
  - Send alerts for failures or stop from sending the data to downstream
  - DAG design: start → check parquet schema → (if okay) load to bigquery → (if not okay) block ingestion → stores results → sends alerts → end

Data Monitoring

Custom rules can be created using a YAML file and then it will be compiled to the Airflow DAG. The anatomy of DQ configs would look like this for example:

```
table_name: "transactions"
dataset: "tmt-dw.production"
description: "Accuracy and validity rules for transactions table"

rule_categories:
  accuracy_rules:

    transaction_amount_accuracy:
      rule_type: "range_validation"
      description: "Transaction amounts should be within realistic range"
      query_template: |
        SELECT
          COUNT(*) as total_records,
          COUNTIF(amount < 1000 OR amount > 1000000) as invalid_count,
          ROUND(COUNTIF(amount < 1000 OR amount > 1000000) / COUNT(*) *
100, 2) as invalid_percentage
        FROM `{dataset}.{table_name}`
        WHERE business_date = CURRENT_DATE('Asia/Jakarta')
      thresholds:
        max_invalid_percentage: 1.0
        severity: "critical"

  validity_rules:

    transaction_id_format:
      rule_type: "format_validation"
      description: "Transaction IDs should follow TXN-YYYYMMDD-XXXXXX
format"
      query_template: |
        SELECT
          COUNT(*) as total_records,
          COUNTIF(NOT REGEXP_CONTAINS(transaction_id,
r'^TXN-\d{8}-\d{6}$')) as invalid_format_count,
          ROUND(COUNTIF(NOT REGEXP_CONTAINS(transaction_id,
r'^TXN-\d{8}-\d{6}$')) / COUNT(*) * 100, 2) as invalid_percentage
        FROM `{dataset}.{table_name}`
        WHERE business_date = CURRENT_DATE('Asia/Jakarta')
      thresholds:
        max_invalid_percentage: 0.0
        severity: "critical"

execution_schedule:
  frequency: "daily"
  run_time: "08:00"
  timezone: "Asia/Jakarta"
  <or specify cron expression like * 8 * * *>

notification_settings:
  enabled: true
  channels: ["slack", "email"]
```

The anatomy of the config:

1. What table to monitor

```
table_name: "transactions"
dataset: "tmt-dw.production"
```

2. What rules to check

```
rule_categories:
  accuracy_rules:
    transaction_amount_accuracy:
      query_template: "SELECT COUNT(*) FROM table WHERE amount > 1000000"
      thresholds:
        max_invalid_percentage: 1.0
        severity: "high"
```

3. When to run

```
execution_schedule:
  frequency: "daily"
  run_time: "08:00"
```

4. How to alert

```
notification_settings:
  channels: ["slack", "email"]
```

And it can be translated to a Airflow DAG: dq_monitor_transactions_tmt_dw_production that runs daily every 8 AM (* 8 * * *).

Data Alerting

Real-time notification system for data quality issues.

For alerting we need modularity to select channels, having reusable message templates, and choosing severity. For that we can store our alert templates data on a table, would look like this:

```
CREATE TABLE alert_templates (
    alert_id STRING,            -- 'FRESHNESS_VIOLATION_CRITICAL'
    title_template STRING,      -- '🚨 {table_name} Data Missing'
    message_template STRING,    -- 'Table: {table_name}, Delay:
{delay_minutes} mins'
    channels ARRAY<STRING>,     -- ['slack', 'email', 'telegram']
    severity STRING             -- 'critical'
)
```

And for the channel services:

SlackAlertChannel: Webhook with color coding
EmailAlertChannel: SMTP with HTML formatting
TelegramAlertChannel: Bot API with emojis

For the templates:

- Freshness violation

```
alert_id: 'FRESHNESS_VIOLATION_CRITICAL'
message_template: '📊 **Table**: {table_name}\n⏰ **Expected**:
{expected_time}\n⌛ **Delay**: {delay_minutes} minutes'
```

- Volume anomaly

```
alert_id: 'VOLUME_ANOMALY_HIGH'
message_template: '📈 **Current**: {current_volume:,}\n📊 **Expected**:
{baseline_average:,}\n📈 **Deviation**: {deviation_percentage}%'
```

- Rule violation

```
alert_id: 'DATA_QUALITY_RULE_VIOLATION'
message_template: '🔍 **Rule**: {rule_name}\n❌ **Issue**:
{violation_details}\n🎯 **Threshold**: {threshold}'
```

And on the integration we can do following pattern:

```
# Infreshness DAG
if violation_detected:
    send_alert('FRESHNESS_VIOLATION_CRITICAL', violation_data)


# In volume DAG
if anomaly_detected:
    send_alert('VOLUME_ANOMALY_HIGH', anomaly_data)


# In data quality monitoring
if rule_failed:
    send_alert('DATA_QUALITY_RULE_VIOLATION', rule_data)
```

Tableau

Core Concept: Pre-aggregated DQ Scores & Dimensions
Instead of Tableau querying raw monitoring data, create business-friendly data marts with:

- Overall DQ scores (0-100) per table
- Individual metric scores (accuracy, completeness, timeliness, etc.)
- Grades (A, B, C, D, F) for easy interpretation
- Trends and alerts pre-calculated

For the dashboard, we can specify for needs:

Executive Dashboard
- Portfolio Health: Average scores, grade distribution
- Business Impact: Critical tables at risk
- Trend Analysis: Improvement rates over time

Operational Dashboard
- Real-time Scorecard: Table scores
- Alert Status: SLA violations with ✅❌🚨 indicators
- Dimension Heatmap: Color-coded performance matrix

And to generate this, we can use separate Airflow DAG for data mart refresh:
1. Calculate dimension scores from raw monitoring data
2. Compute overall DQ scores (weighted average)
3. Generate grades and health status

End-to-end Flow

```
Data Arrival → Freshness → Schema → Load → Volume → Rules → Mart →
Tableau
```

How The System Work Together

Data Quality Pipeline Integration:
1. Data Lands in GCS → Freshness Monitor (immediate check)
2. If Fresh → Schema Evolution Check (before BigQuery load)
3. If Schema OK → Load to BigQuery → Volume Anomaly Check
4. If Volume Normal → Completeness & Rule Validation
5. All Results → Data Mart → Tableau Dashboards

Cross-System Dependencies:
- Schema blocking stops entire pipeline
- Freshness failure delays downstream checks
- All systems feed into unified alerting
- Data mart aggregates all quality dimensions

How Rules Get Deployed

Rule Deployment Workflow:

1. Rule Creation:
    - User creates YAML rule file
    - Commits to Git repository
    - Triggers CI/CD pipeline

2. Validation Process:
    - Validation_pipeline:
        - Syntax check (YAML validity)
        - SQL query validation

3. Deployment Stages:
    a. Development:
        - Deploy to dev environment
        - Run against test data
        - Validate rule logic

    b. Staging:
        - Run against production-like data
        - Performance testing

    c. Production:
        - Deploy during maintenance window
        - Immediate monitoring for new rule

5. Rule Lifecycle Management:
    - Version tracking in Git
    - Rule performance monitoring
    - Sunset inactive/ineffective rules (rationalization)