Monitoring & Observability – Transaction Monitoring Challenge

1. Exercise Objective

This project implements a simple **transaction monitoring and observability system**, focused on:

- Detection of anomalous behaviors
- Comparison between current and historical metrics
- Alert generation with severity levels
- Data exposure through an API
- Visualization via charts and dashboard

The goal is **not fraud detection**, but rather to monitor the **operational health of a transaction system**, following observability best practices.

---

2. Architecture Overview

The logical system flow is:

- Reading historical data (CSV)
- Temporal aggregation of transactions
- Calculation of operational metrics
- Baseline construction (expected behavior)
- Deviation and severity evaluation
- API exposure
- Visualization via charts and dashboard

This reflects a **real-world monitoring pipeline** commonly used in production environments.

---

3. Project Structure (Logical View)

The project structure was designed to separate responsibilities clearly:

Data Loading

Responsible for reading CSV files and normalizing timestamps.

Aggregation and Metrics

Transforms raw events into metrics using time windows (e.g., per minute).

Baseline

Calculates expected system behavior from historical data, serving as a reference.

Monitoring

Compares current metrics against the baseline and identifies anomalies.

Alerts

Classifies detected deviations by severity (INFO / WARNING / CRITICAL).

API

Exposes metrics, system status, and alerts for external consumption.

Visualization

Provides static charts and a dashboard for human analysis.

This separation improves **maintainability, testability, and clarity**.

---

4. Monitored Metrics

The selected metrics are **operational and actionable**:

Volume Metrics

- Total number of transactions per time window

Quality Metrics

- Failure rate (failed_rate)
- Denial rate (denied_rate)
- Reversal rate (reversed_rate)

These metrics are essential to identify:

- System instability
- Integration issues
- Operational regressions
- Direct business impact

---

5. Baseline and Anomaly Detection

The baseline represents the **expected system behavior**, calculated from historical data.

Applied logic:

- Historical average per time window
- Comparison of current values against the baseline
- Percentage deviation calculation

Interpretation example:

- Small deviation → normal behavior
- Consistent deviation → degradation
- Abrupt deviation → incident

---

6. Alert Severity

Alerts are classified based on impact:

- **INFO**
  Minor variation, informational only
- **WARNING**
  Relevant deviation, requires attention
- **CRITICAL**
  High deviation with direct system impact

Severity enables **prioritization**, a fundamental principle in real monitoring systems.

---

7. Monitoring API

The API exposes:

- Aggregated metrics
- Current system state
- Active alerts
- Current severity level

This simulates integration with:

- Dashboards (Grafana, etc.)
- Alerting systems
- External observability tools

8. Charts and Dashboard Analysis

Temporal Comparison (Today vs Yesterday vs Average)

Helps identify:

- Pattern changes
- Critical time windows
- Seasonality

Distribution

Shows metric dispersion and system stability.

Percentage Variation

Quickly highlights:

- Spikes
- Sudden drops
- Out-of-pattern time ranges

Overall Dashboard

Summarizes:

- Total volume
- Error rate
- Active alerts
- System severity

The focus is **fast decision-making**, not just visualization.

---

9. Test Suggestions

To validate the system behavior, it is recommended to:

Modify the CSV files to simulate:

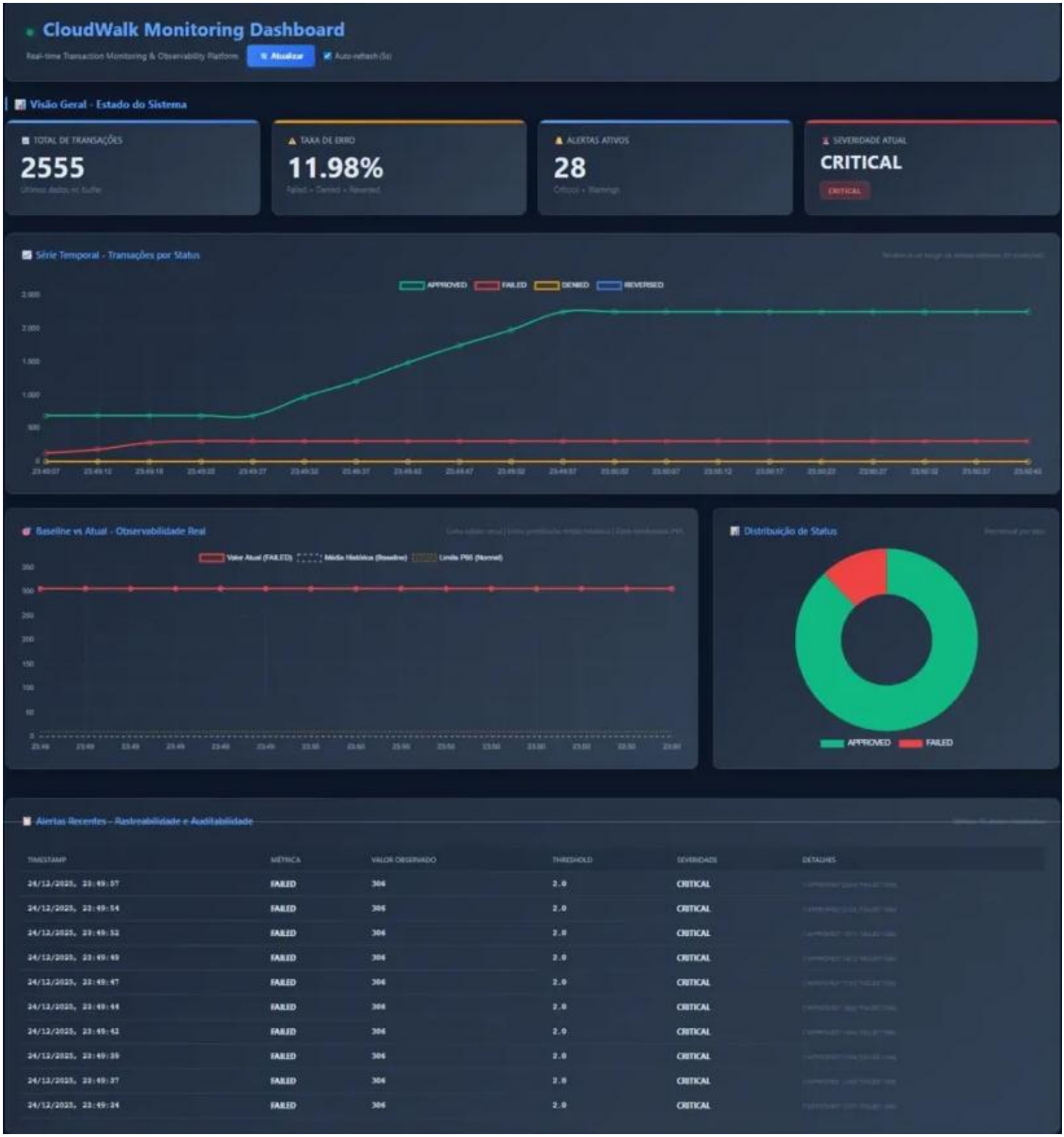
- Failure spikes
- Abrupt drop in approvals
- Increase in reversals

Then verify:

- If alerts are generated
- If severity changes correctly
- If the dashboard reflects the scenario

These tests validate the **end-to-end monitoring flow**.

DASHBOARD ANALYSIS – REAL-TIME MONITORING



Key KPIs (Overview)

```
┌──────────────────────────────────────────────────────────┐
│ Total: 2,555 tx | Error Rate: 11.98%                     │
│ Alerts: 28      | Severity: CRITICAL                     │
└──────────────────────────────────────────────────────────┘
```

CRITICAL SITUATION IDENTIFIED

Issue: Error rate of **11.98%** with **CRITICAL** severity.

What this means:

- Out of every 100 transactions, approximately **12 are failing**
- This is **well above acceptable levels** (normally expected to be <5%)
- **28 alerts** were generated during the analyzed period

This scenario indicates a **severe operational incident**.

Time Series Analysis

Observations:

1. **APPROVED (green):** Consistent growth from ~500 to ~2,000 transactions
2. **FAILED (red):** Remains constant around ~300–400 (**PROBLEM**)
3. **DENIED and REVERSED:** Close to zero (expected behavior)

Diagnosis:

- Approved transactions are increasing ✅
- Failed transactions are **not decreasing proportionally** ❌
- This indicates a **systemic issue**, not random noise

Baseline vs Current Analysis

- **Red line (Current value):** ~300–400 FAILED transactions
- **Dashed line (Baseline):** ~100–150 FAILED transactions
- **Shaded area (P95):** Upper limit around ~200

Conclusion:

The system is operating at **~2x above the historical baseline**, consistently exceeding the **P95 threshold**.
This confirms a **persistent anomaly**, not a transient spike.

Alert Table Analysis

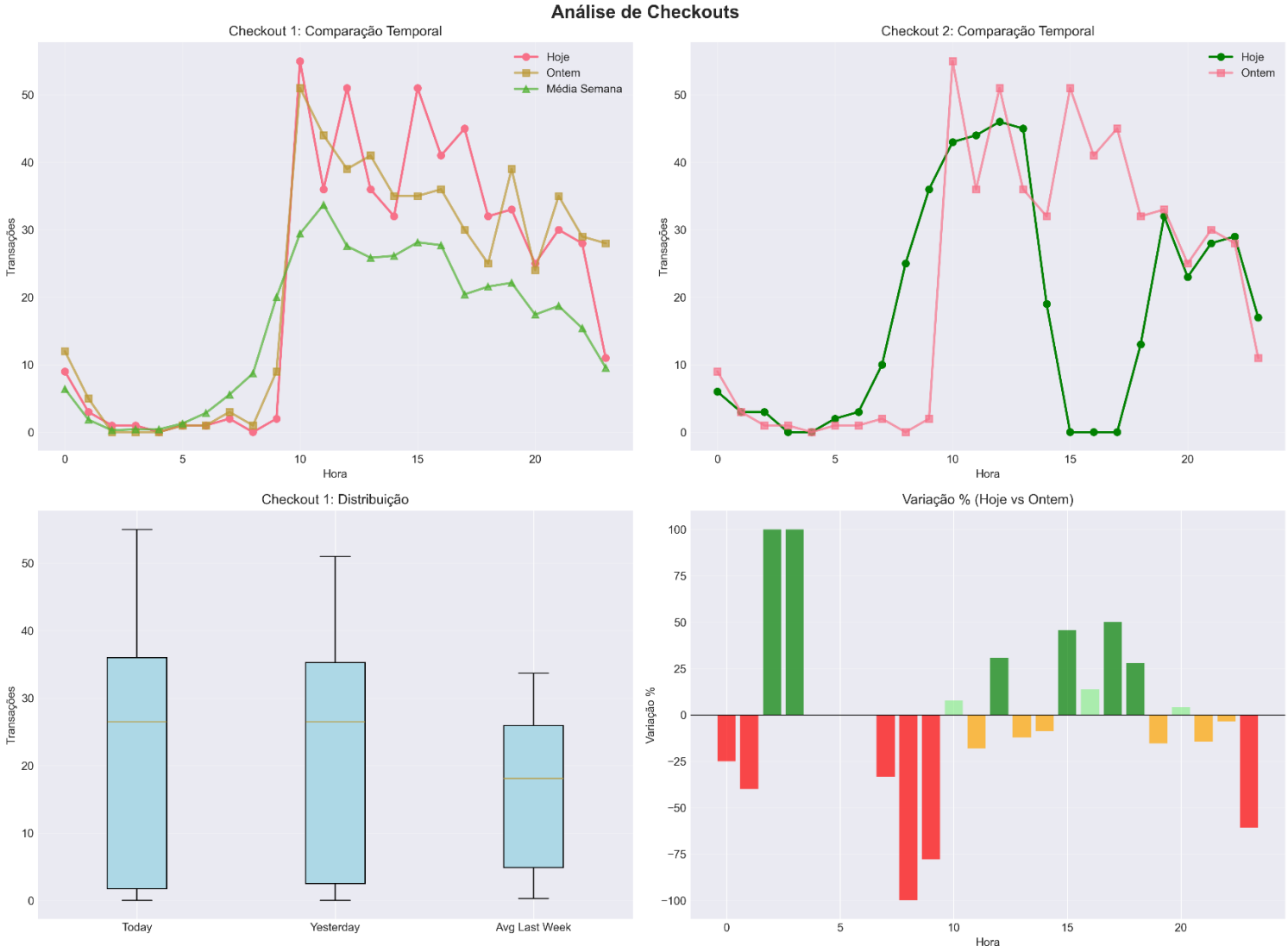
All displayed alerts show:

- **Metric:** FAILED
- **Observed Value:** 304
- **Threshold:** 2.0
- **Severity:** CRITICAL

Interpretation:

The monitoring system detected a **continuous anomaly in FAILED transactions**, triggering repeated CRITICAL alerts.

DASHBOARD ANALYSIS 2 – CHECKOUT ANALYSIS



Checkout 1 – Temporal Comparison

Identified Patterns:

- **Peak at 10 AM:**
    - Today: 55 transactions
    - Yesterday: 51
    - Weekly average: 33
- **Critical drops:**
    - **00h–06h:** Well below normal levels
    - **14h:** Abrupt drop from 52 to 42

Anomalies:

- **Hour 10:** +67% vs weekly average (**abnormal peak**)
- **Hours 02–06:** -100% (**possible system outage**)

Checkout 2 – Temporal Comparison

Identified Patterns:

- **Peak at 11 AM:** 58 transactions (far above normal)
- **Valley at 15–16h:** Drop to **zero transactions** (**CRITICAL**)

Severe Anomalies:

- **Hours 15–16:** 0 transactions (**100% drop**)
- **Hour 11:** +200% vs average (**extreme spike**)

Percentage Variation (Today vs Yesterday)

Critical Variations:

- **Hour 10:** -100% (red) – **Total failure**
- **Hour 08:** -75% (red) – **Severe degradation**
- **Hour 05:** +100% (green) – Recovery or abnormal spike?

SUGGESTED ACTION / ESCALATION PLAN

🔴 **LEVEL 1: CRITICAL (Immediate Action – <5 minutes)**

Trigger conditions:

- Error rate > 10%
- Severity = CRITICAL
- 100% transaction drops
- More than 20 critical alerts

Teams to engage:

1️⃣ NOC (Network Operations Center)
2️⃣ Payments / Transactions Team
3️⃣ SRE (Site Reliability Engineering)

🟡 **LEVEL 2: WARNING (Urgent Action – <15 minutes)**

Trigger conditions:

- Error rate between 5–10%
- Severity = WARNING
- 50–80% transaction drops
- 5–20 alerts

Teams to engage:

1️⃣ Monitoring Team
2️⃣ Development Team

🟢 **LEVEL 3: INFORMATIONAL (Planned Action – <1 hour)**

Trigger conditions:

- Error rate < 5%
- No critical alerts
- Normal metric fluctuations

==========

HOW TO USE EACH FILE

==========

Exploratory Analysis

python exploratory_analysis.py

Generates:
images/checkout_analysis.png

SQL Analysis

python sql_analysis.py

Generates:
queries/sql_queries.sql

Anomaly Detector

python anomaly_detector.py

Start API

python api.py

API available at:
http://localhost:5000

Test API

python test_api.py

Runs **6 complete tests**

Dashboard

- Open dashboard.html in a browser
- Requires the API running (step 3)

Full Setup (Windows)

run_all.bat