International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Impact Factor 2024: 7.101

Automating Performance Degradation Detection in CI/CD Pipelines

Alex Kuriakose

Senior Software Engineer, Workday Inc.

Abstract: ***In production settings, deteriorating performance can result in lost revenue, higher infrastructure expenses, and a bad user experience. Automating performance validation is crucial for identifying and averting regressions early in the development lifecycle in contemporary CI/CD pipelines. This journal describes a methodical approach to determine response time degradation, specify pass/fail criteria, and integrate best practices for a robust performance analysis. To increase the precision of performance validation in CI/CD pipelines, the suggested methodology makes use of weighted scoring, statistical filtering, trend detection, and dynamic baselines.***

Keywords: Performance Degradation Detection, CI/CD Pipeline Automation, Response Time Analysis, Weighted Scoring Methodology, Dynamic Baselines for Performance Validation

1. Introduction

Background and Motivation

It is essential to incorporate a strong performance evaluation component into the CI/CD pipeline as part of the **universal shift - left methodology**. By identifying performance bottlenecks early in the development lifecycle, shift - left testing significantly lowers the cost of fixing these problems later.

According to industry research, the cost of fixing performance issues increases exponentially as the defect moves from development to production. Early detection allows development teams to fix bottlenecks before deployment, reducing downtime, infrastructure costs, and user impact. For example, a 100 - millisecond increase in page load time can reduce conversion rates by **7%** and raise infrastructure costs by **30%** due to increased server load. Performance regressions are not only costly in terms of infrastructure but can also impact user retention, search rankings, and overall business revenue.

In such a scenario, the **algorithm used to evaluate performance degradation** becomes critical. A well - designed algorithm should be able to:

- Accurately detect performance degradation
- Weigh critical transactions appropriately
- Handle outliers and noise
- Avoid false positives and false negatives
- Allow configurability to accommodate different application types and load patterns

Making the algorithm adaptable to various application domains, load patterns, and application sizes is another crucial aspect. It is highly unlikely that a one - size - fits - all strategy would work on large scale, complex systems.

2. Problem Statement

Traditional performance validation in CI/CD pipelines faces several challenges:

- Simple threshold - based validation often leads to false positives or negatives.
- Lack of context around business - critical endpoints can misrepresent the severity of degradation.
- Outliers and noise from environmental factors can skew results.
- Additionally, gradual degradation over multiple releases may go unnoticed.
- Inconsistent test outcomes due to cold starts or initial warmup noise.

An effective solution should:

- Provide a structured scoring mechanism based on response time changes.
- Account for endpoint criticality through weighted scoring.
- Filter out statistical outliers.
- Detect trends over multiple releases.
- Offer clear and consistent pass/fail criteria.
- Adapt dynamically to different load patterns and application sizes.

3. Methodology

1) Response Time Degradation Formula

The response time degradation score (S) is computed as:

$$S = \frac{Rn - Ro}{Ro} * 100$$

where:

- **S** = Response time degradation score (%)
- Rn = Average response time for the new release
- R_o = Average response time for the previous release

2) Severity Classification

Based on the computed degradation score, performance outcomes are classified into three zones:

Zone	Criteria	Interpretation
Green	S≤+5%	Minimal or no degradation — Acceptable
Yellow	+5%<S≤+20%	Moderate degradation — Warning
Red	S>+20%	Significant degradation — Critical

3) Baseline and Tolerance Ranges

Rather than using static thresholds, acceptable deviation is adjusted dynamically based on historical performance. A rolling average over the last 5–10 releases define the baseline:
Tolerance = Baseline * (1 + acceptable deviation)
Example:

- Baseline = 200 ms
- Acceptable deviation = 5%

➡️ Acceptable range = 200 ms ± 10 ms
This ensures that gradual performance improvements or acceptable fluctuations are handled gracefully.

4) Pass/Fail Criteria
The pass/fail logic incorporates two key conditions:

- Any red transaction = FAIL
- More than 50% yellow transactions = FAIL

If neither of the above occurs, the result is a PASS.

5) Gray Zone Handling
To prevent hard failures on borderline cases, a "Gray Zone" is introduced:

- If yellow transaction percentage is exactly 50%, the result is marked as "Inconclusive. "
- In such cases, the test is either manually reviewed or a re - test is triggered

6) Weighted Scoring
Endpoints are assigned weights based on their business criticality to reflect the actual impact of performance degradation. The weighted score is computed as:

$$S_{weighted} = \frac{\sum_{i=1}^{n} Wi \cdot Si}{\sum_{i=1}^{n} Wi}$$

where:

- w_i = Weight based on criticality
- S_i = Degradation score for each transaction

Criticality	Weight	Example Endpoints
Critical	1.5	Checkout, Login
High	1.2	Product Search
Normal	1	Category Browsing
Low	0.8	Static Content

7) Outlier Detection
Outliers caused by temporary infrastructure issues or network spikes are filtered out using the 3 - sigma rule:

- Compute the mean (μ) and standard deviation (σ) of the scores.
- Remove scores that deviate from the mean by more than 3σ:

$\mu - 3\sigma \leq S \leq \mu + 3\sigma$

8) Trend Detection
Degradation over multiple releases may indicate a gradual regression even if individual scores pass the criteria. A release will fail if degradation persists across the last three or more consecutive releases:
$S_{n-3} > 0, S_{n-2} > 0, S_{n-1} > 0$

9) Separate Warmup and Steady - State Metrics

- Initial warmup requests often show higher response times. To prevent skewing, separate warmup from steady - state performance:
- First 5–10% of transactions are excluded from analysis.

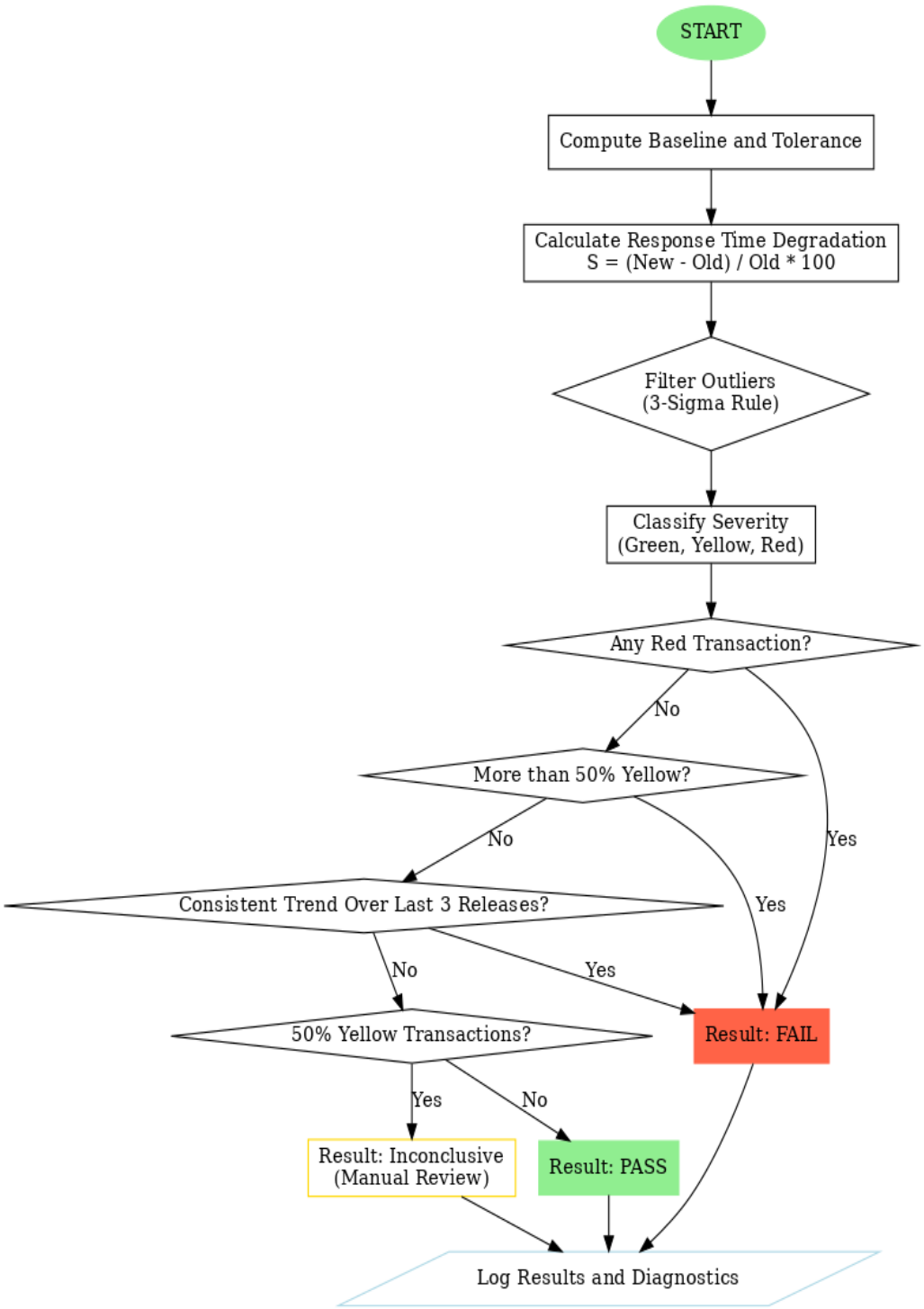
10)Graceful Handling of Missing Data

- Transactions with invalid or missing response time data are ignored.
- If over 10% of the data is missing, the test is flagged as inconclusive.

11)Logging and Diagnostics
Detailed logging is added to simplify root cause analysis:

- Capture failed transactions and scores.
- Include timestamps and endpoint names.
- Generate diagnostic reports for analysis.

Implementation
The methodology was implemented using Python and integrated into a CI/CD pipeline. The system dynamically adjusts thresholds based on historical data and transaction weighting, ensuring accurate performance validation across diverse load conditions.



4. Challenges and Limitations

- Weight tuning requires ongoing adjustment based on business feedback.
- Trend analysis may miss performance drops if they are not sustained.
- Outlier filtering can remove valid data if sample size is small.

5. Conclusion

The proposed solution provides a structured, adaptive approach to performance validation in CI/CD pipelines. By combining weighted scoring, outlier removal, and trend analysis, it enables more accurate pass/fail decisions and helps identify degradation patterns early in the development cycle.

References

[1] **Fowler, M. (2018).** *Continuous Integration and Continuous Deployment (CI/CD).* ThoughtWorks.
[2] **Shahin, M., Ali Babar, M., & Zhu, L. (2017).** Continuous Integration, Delivery, and Deployment: A Systematic Review on Approaches, Tools, Challenges, and Practices. *IEEE Access, 5*, 3909–3943.
[3] **Apache JMeter. (2023).** *The Apache JMeter Project.* Retrieved from https: //jmeter. apache. org

[4] **Docker Inc. (2023).** *Docker Documentation.* Retrieved from https: //docs. docker. com

[5] **PostgreSQL Global Development Group. (2023).** *PostgreSQL: The World's Most Advanced Open Source Relational Database.* Retrieved from https: //www.postgresql. org

[6] **Grafana Labs. (2023).** *Grafana: The Open Observability Platform.* Retrieved from https: //grafana. com

[7] **Chen, L., & Babar, M. A. (2014).** A systematic review of evaluation of variability management approaches in software product lines. *Information and Software Technology, 55 (5),* 705 - 725.

[8] **Breck, E., Cai, S., Nielsen, E., Salib, M., & Sculley, D. (2019).** The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction. *IEEE Big Data Conference Proceedings.*

[9] **Roy, S., & Cordy, J. R. (2020).** A Survey on Automated Performance Testing Approaches. *ACM Computing Surveys, 53 (5),* 1 - 36.

[10] **Brunnert, A., Vögele, C., Krcmar, H. (2018).** Performance Testing in Continuous Delivery: A Case Study on Automation. *Journal of Systems and Software, 145,* 141 - 153.