nagarro

# Goveranance first: Performance test orchestration

A pull-based execution model with automated validation and reporting

nagarro

## Table of Contents


| | |
|---|---|
| Introduction | 01 |
| Challenges | 02 |
| Technology stack | 03 |
| Specification-driven design before code | 04 |
| Governance constraints | 05 |
| Architecture overview | 06 |
| Security posture | 07 |
| Implementation summary | 08 |
| Same-day multi cycle test comparison | 09 |
| Evidence | 10 |
| Business and engineering impact | 13 |
| Lessons learned from AI-assisted development | 14 |
| Scope limits and non-goals | 15 |
| Conclusion | 15 |
| About the author | 16 |
| Reference | 16 |

# Introduction

This white paper presents a production- oriented performance test orchestration system built through GitHub Copilot Specification Driven Development (SDD), where architecture, controls, and governance are defined in structured specifications before any code is written.




©2026 Nagarro

# Challenges

Performance testing at enterprise scale is effort-intensive. A single test cycle from VM login to report delivery takes several hours. Results vary by engineer. There is no standard validation, no consistent report format, and no auditable trail. This system was built to fix that. The pipeline runs JMeter on the remote VM, validates results, and delivers a structured report to Slack. Delivery time varies by complexity. A single run lands in minutes. A multi-cycle comparison is completed within the same session.

The system uses a pull-based model: a local machine submits run requests, a remote VM runner claims and executes Apache JMeter tests, and validated lifecycle reports are delivered to Slack. The VM environment stores no delivery credentials and exposes no inbound orchestration endpoint. The primary contribution is a governance-first delivery method that improves consistency, traceability, and security posture while reducing manual handoffs. Two reliability safeguards in the comparator, a measurement noise floor and a load profile compatibility guard, prevent millisecond-level oscillations and mismatched sample counts from surfacing as false regressions or improvements.

Remote performance testing in many enterprise teams is still operationally manual: log into a VM, run JMeter, collect output files, and post a summary in a chat channel. This works on a small scale but breaks under audit pressure and release-critical timelines.

Four recurring issues surface:

- **Inconsistent validation**: Pass or fail outcomes vary by engineer when threshold logic is not standardized.
- **Unstructured reporting**: Different run summaries use different formats and metric focus, making comparison unreliable.
- **Weak traceability**: When regressions surface later, teams struggle to connect request intent, execution state, raw artifacts, and stakeholder report in one auditable chain.
- **Noisy comparisons**: Naive percentage deltas misclassify normal measurement noise as regression when latency sits in second-digit milliseconds, and mismatched sample counts between runs can produce throughput swings that look real but aren't.


©2026 Nagarro

# Technology stack

| Layer | Technology | Purpose |
|---|---|---|
| Orchestration and Execution | Python 3.11 | Request handling, JMeter invocation, parsing, validation, comparison, report assembly |
| Notification Delivery | TypeScript 5.6 | Slack message formatting and HTTP delivery |
| Performance Test Tool | Apache JMeter | Load generation and result capture on VM |
| Notification Channel | Slack Incoming Webhooks | Lifecycle events and final report delivery |
| Scripting and Automation | PowerShell | Local submit and VM runner startup |
| Design Framework | GitHub Copilot SDD | Specification-driven design and implementation governance |

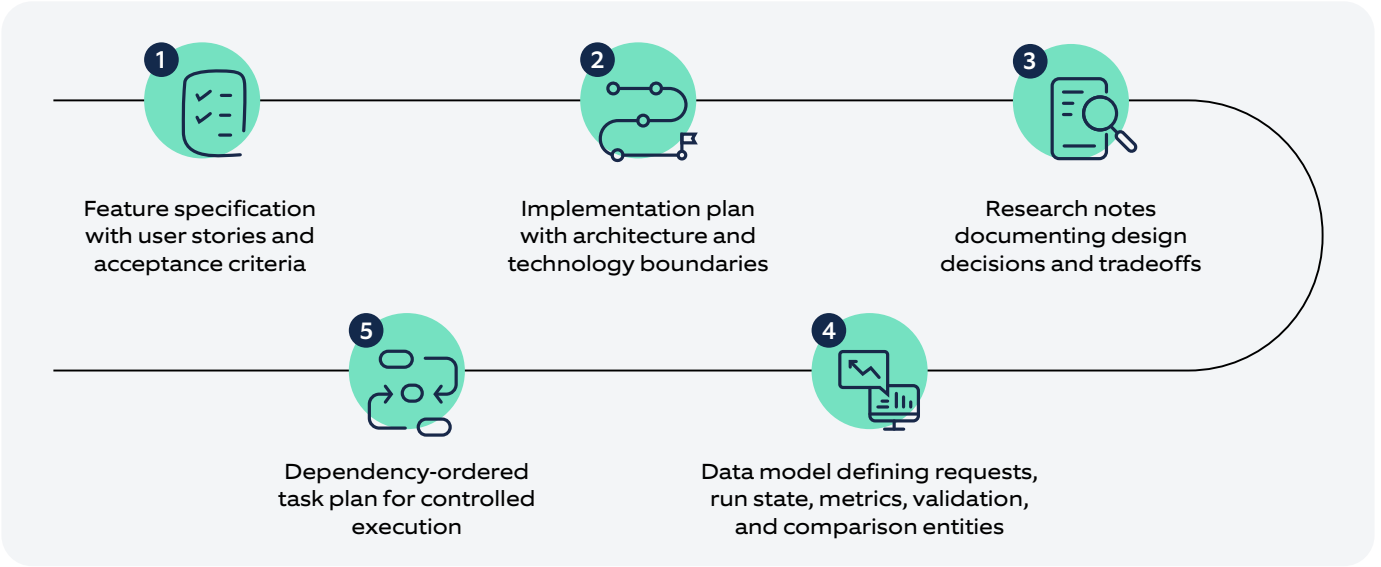
This release targets JMeter first. The architecture accepts future adapters for tools such as k6 and Gatling without redesigning the full orchestration model.


©2026 Nagarro

# Specification-driven design before code

Add this paragraph: In AI-assisted development, the main risk is building code on wrong assumptions. Without defining the constraints upfront, AI fills the gaps on its own. Those gaps then show up in security handling, credential management, and validation logic. Catching them late costs more than preventing them early.

No code was written until the full specification chain was complete:




This sequence reduced ambiguity before coding started and produced a documented decision trail available for future systems.


©2026 Nagarro

# **Governance constraints** (Constitution)

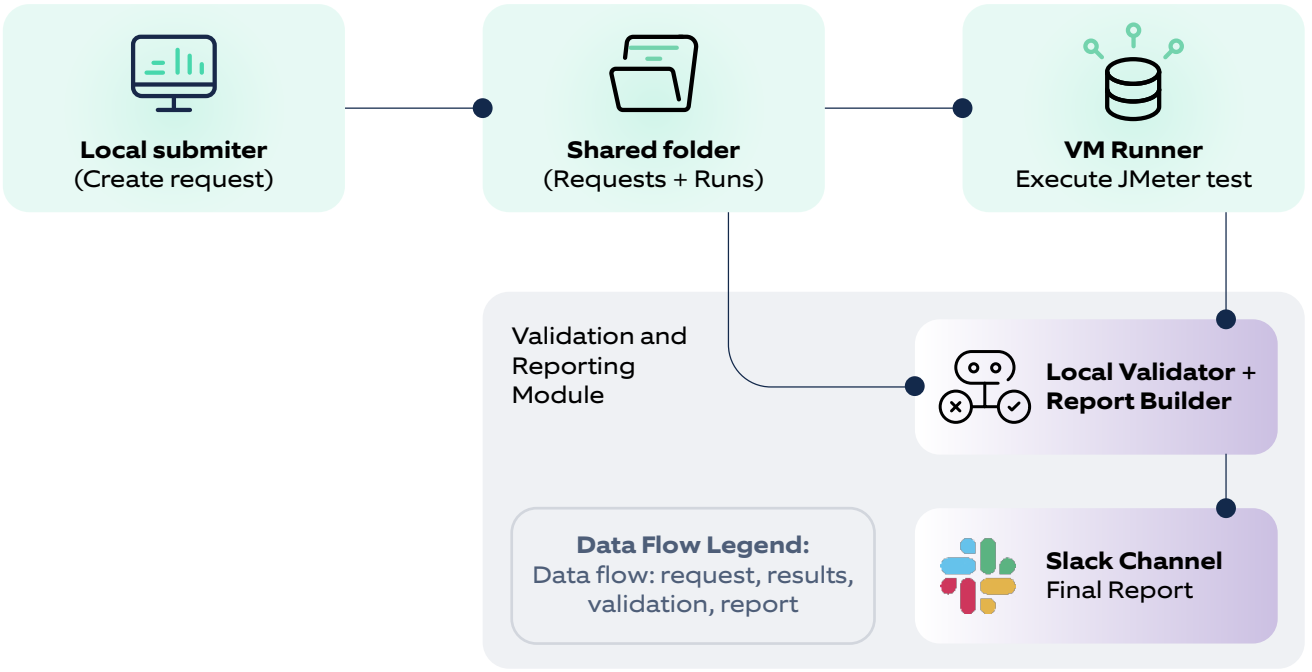
A dedicated governance layer defined nonnegotiable rules before development:

- Validation must be completed and its outcome must be embedded in every final report. Validation is advisory: a failed check annotates the report and the lifecycle event but does not suppress publication. We rejected report suppression because release decisions require evidence even when targets are missing.
- Slack webhook values must come from runtime environment configuration.
- Secrets and credentials must not be stored in repository artifacts.
- VM execution must follow shared-folder pull orchestration.
- VM operation must not depend on inbound control endpoints or repository credentials.

Treating these as architecture constraints, not coding preferences, prevented security drift and late-stage rework.


©2026 Nagarro

# Architecture overview




- The solution uses a pull-based shared folder orchestration pattern.

### High-level flow

- Local side validates requests and writes a run manifest.
- VM runner polls queue location and claim work.
- VM executes JMeter and writes status, logs, and results to run folders.
- Local side reads outputs, performs validation and optional comparison and sends the final stakeholder report.

### Why pull instead of push

Push models require local orchestration to invoke remote execution through SSH, APIs or agents, increasing credential footprint and endpoint exposure.

In the pull model used here, remote execution claims work from a shared contract, reducing direct control dependencies while preserving observability.


©2026 Nagarro

# Security posture

The VM execution footprint is minimal.

**VM contains**

- JMeter binaries
- Runner package
- Access to shared execution path

**VM does not contain**

- Git repository clone requirement
- GitHub credentials
- Slack webhook secrets
- Inbound orchestration API endpoint requirement

Runner deployment is package-based and controlled. Keeping notification delivery on the local side reduces credential spread and limits the blast radius if the execution host is compromised.




©2026 Nagarro

# Implementation summary

The production flow is structured and deterministic:

- Local orchestrator accepts requests and initializes run structure.
- VM runner polls and claims request.
- JMeter executes and produces artifacts.
- Parser extracts key metrics from JTL output: average response, P95, P99, throughput, error rate.
- Validator applies thresholds and produces explicit pass or fail reasons.
- Comparison logic executes when multi-run input is present.
- Report builders create structured output.
- Slack notifier sends approved lifecycle and final report messages.

### Default validation thresholds

| Check | Default Threshold | What a Failure Indicates |
|---|---|---|
| Error rate | ≤ 1.00% | Too many requests failed. |
| P95 response time | ≤ 2000 ms | Most users are seeing slow responses. |
| P99 response time | ≤ 3000 ms | Worst-case requests are too slow. |
| Throughput | > 0 req/sec | The test produced no load, likely misconfigured |

Each failed check includes a plain-english explanation for fast triage.

**Comparator reliability safeguards**. Every comparison runs two checks before classification:

- **Noise floor (5 ms)** The comparator labels latency changes smaller than 5 ms as stable, regardless of percentage. A P99 of 3 ms → 4 ms is measurement noise, not a 33% regression.
- **Profile guard (±20% samples)** If two runs differ by more than 20% in sample count, the report flags them as different workloads instead of treating the throughput delta as a real change.


©2026 Nagarro

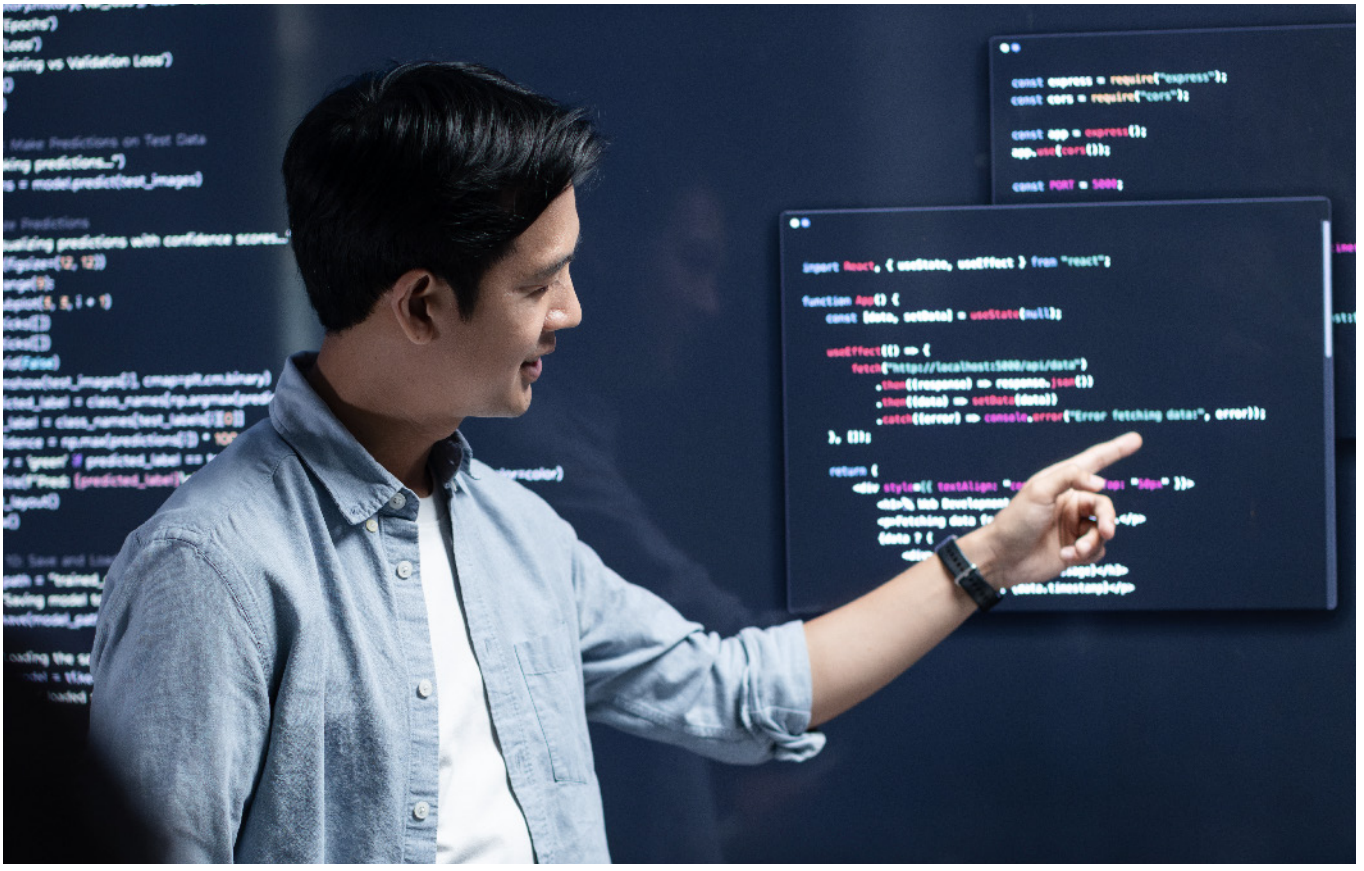
# Same-day multi-cycle test comparison

When two runs are submitted for the same context in one request window, the system executes sequentially, validates each run independently, and then compares outcomes.

Comparison includes:

- Throughput delta analysis
- Latency delta analysis (average, P95, P99)
- Error rate delta analysis
- Classification labels: improved, regressed, stable
- Plain-language verdict for release readiness discussion

The classification stage applies to the 5 ms latency noise floor before assigning labels, and the report header emits a profile compatibility advisory when sample counts between baseline and candidate diverge by more than 20%. Both safeguards are unconditional, appear in the same Slack message engineers already read, and require no operator configuration.

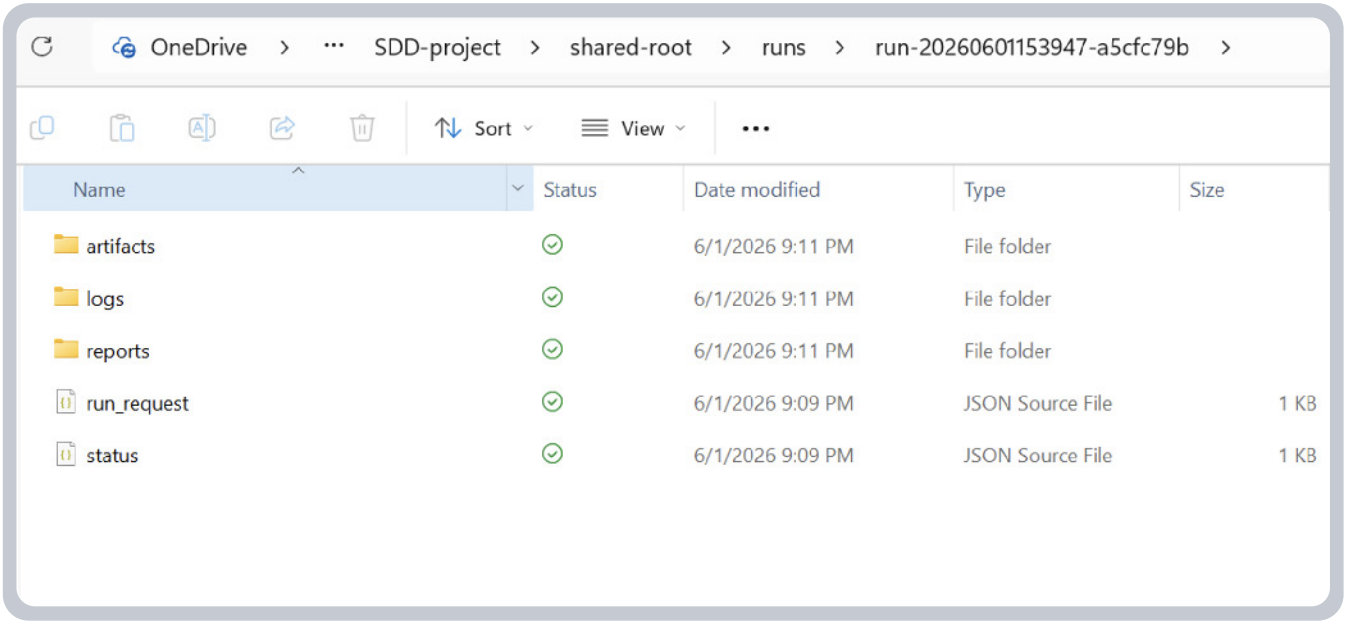



©2026 Nagarro

# Evidence

The figures below illustrate the system end to end: a completed run on disk, the stakeholder-facing Slack report, the specification artifacts produced before code, and the same-day comparison output demonstrating the noise floor and profile guard safeguards.

**Figure 1 — Run traceability structure**

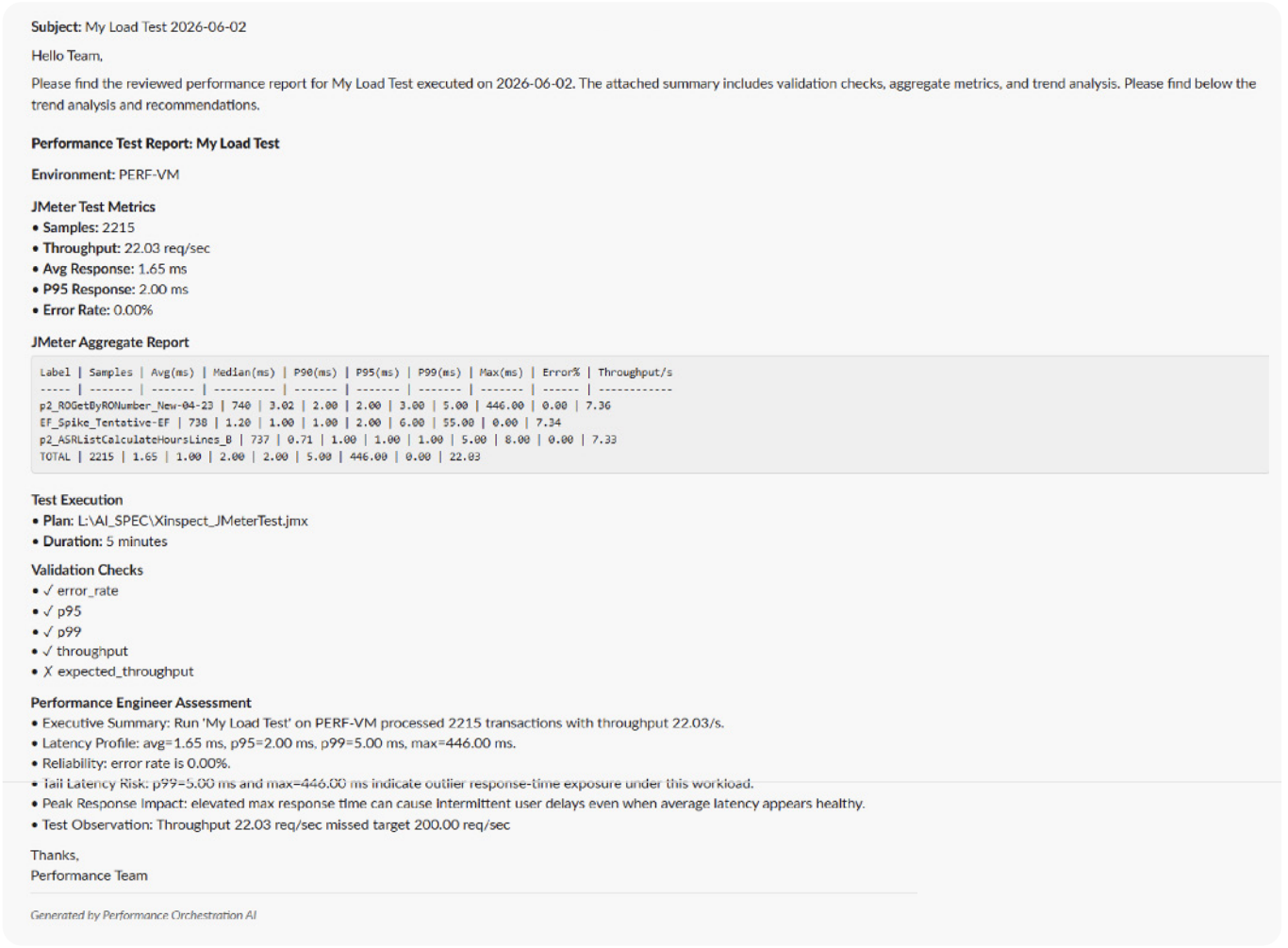



Auditable run artifact chain for one completed execution: request manifest, status file, event log, raw JMeter artifacts, and final report, all under a single run ID.


©2026 Nagarro

# Evidence

**Figure 2 — Final Slack Report (single run)**



**Subject:** My Load Test 2026-06-02

Hello Team,

Please find the reviewed performance report for My Load Test executed on 2026-06-02. The attached summary includes validation checks, aggregate metrics, and trend analysis. Please find below the trend analysis and recommendations.

**Performance Test Report: My Load Test**

**Environment:** PERF-VM

**JMeter Test Metrics**
- **Samples:** 2215
- **Throughput:** 22.03 req/sec
- **Avg Response:** 1.65 ms
- **P95 Response:** 2.00 ms
- **Error Rate:** 0.00%

**JMeter Aggregate Report**

| Label | Samples | Avg(ms) | Median(ms) | P90(ms) | P95(ms) | P99(ms) | Max(ms) | Error% | Throughput/s |
|---|---|---|---|---|---|---|---|---|---|
| p2_ROGetByRONumber_New-04-23 | 740 | 3.02 | 2.00 | 2.00 | 3.00 | 5.00 | 446.00 | 0.00 | 7.36 |
| EF_Spike_Tentative-EF | 738 | 1.20 | 1.00 | 1.00 | 2.00 | 6.00 | 55.00 | 0.00 | 7.34 |
| p2_ASRListCalculateHoursLines_B | 737 | 0.71 | 1.00 | 1.00 | 1.00 | 5.00 | 8.00 | 0.00 | 7.33 |
| TOTAL | 2215 | 1.65 | 1.00 | 2.00 | 2.00 | 5.00 | 446.00 | 0.00 | 22.03 |

**Test Execution**
- **Plan:** L:\AI_SPEC\Xinspect_JMeterTest.jmx
- **Duration:** 5 minutes

**Validation Checks**
- ✓ error_rate
- ✓ p95
- ✓ p99
- ✓ throughput
- ✗ expected_throughput

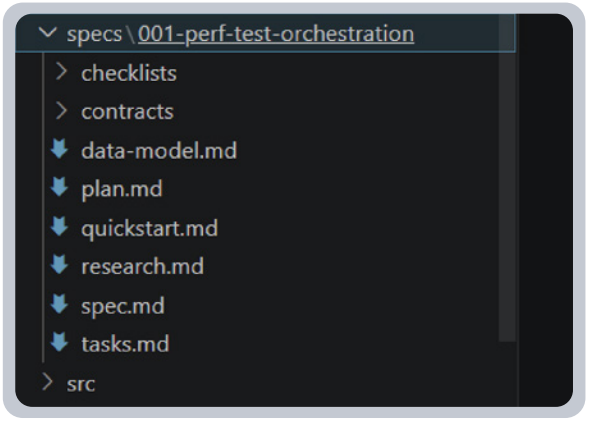
**Performance Engineer Assessment**
- Executive Summary: Run 'My Load Test' on PERF-VM processed 2215 transactions with throughput 22.03/s.
- Latency Profile: avg=1.65 ms, p95=2.00 ms, p99=5.00 ms, max=446.00 ms.
- Reliability: error rate is 0.00%.
- Tail Latency Risk: p99=5.00 ms and max=446.00 ms indicate outlier response-time exposure under this workload.
- Peak Response Impact: elevated max response time can cause intermittent user delays even when average latency appears healthy.
- Test Observation: Throughput 22.03 req/sec missed target 200.00 req/sec

Thanks,
Performance Team

*Generated by Performance Orchestration AI*

Structured Slack final report with validated metrics, JMeter aggregate table, validation checks, and operational observations.

**Figure 3 — SDD artifact chain**

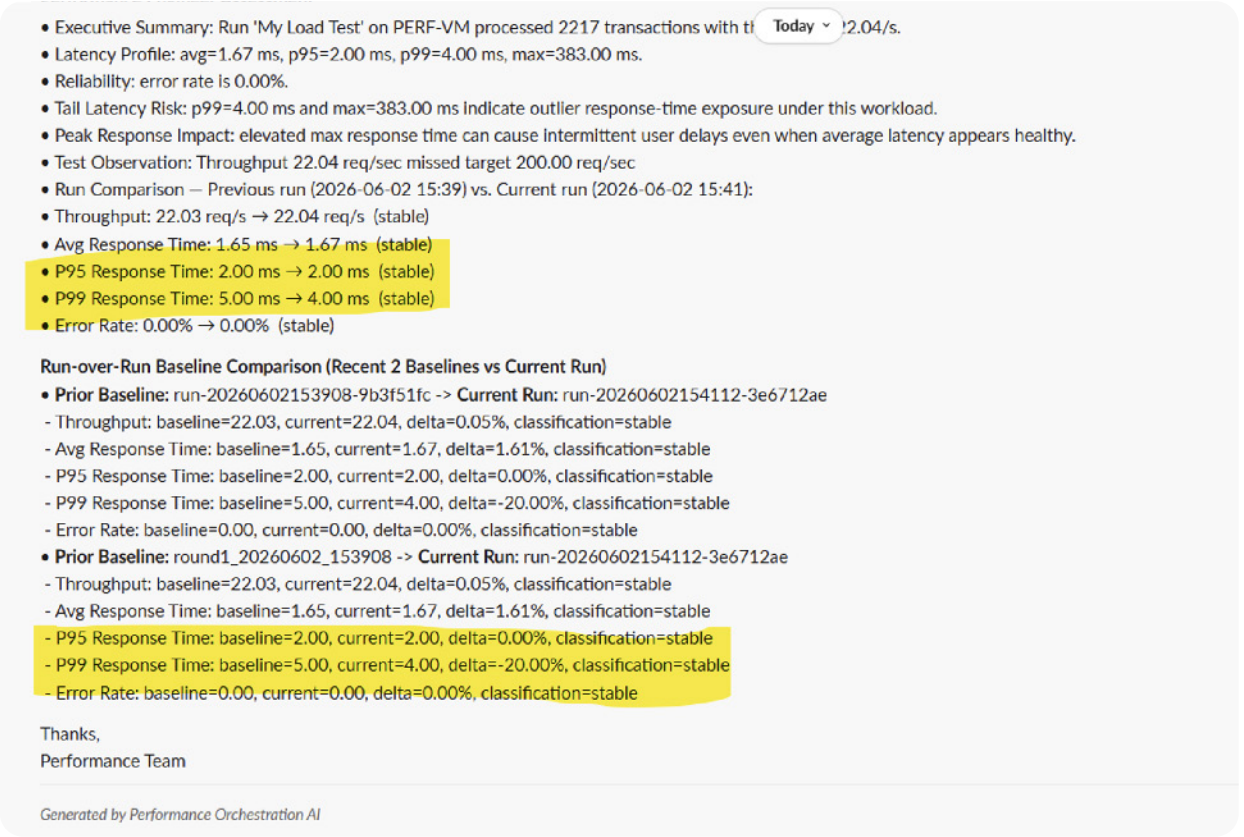



Specification-driven artifact chain (spec, plan, research, data model, contracts, tasks) produced before implementation began.


©2026 Nagarro

# Evidence

**Figure 4 — Same-day multi-cycle comparison with noise floor and profile guard in action**




Two cycles on the same day compared. P99 5 ms → 4 ms (raw −20%) is correctly labelled stable because the absolute change is below the 5 ms noise floor, and no profile advisory is emitted because sample counts (2215 vs 2217) are within the ±20% tolerance. The same panel demonstrates the comparator's standard delta and classification output.


©2026 Nagarro

# Business and engineering impact

The system delivered immediate practical value:

- Reduced manual remote execution and reporting handoffs
- Standardized validation using explicit threshold policy
- Improved auditability through run-level traceable artifacts
- Reduced credential exposure to execution host
- Enabled repeatable cycle over cycle performance assessment
- Removed the two highest frequency false signal classes so stakeholders can act on reports without manual re interpretation. This protects the credibility of the automation, which matters as much as any individual performance metric for a release decision tool.

**Measurement commitment**. Concrete ROI numbers, such as engineer time per run, frequency of manual re interpretation, time to first triage signal, and audit lookup time, will be published in a quantitative appendix once a representative sample has accumulated. Until then, the reliability evidence in Sections 7 to 9 is the defensible claim; cycle time and effort saving figures are not included.





©2026 Nagarro

# Lessons learned from AI-assisted development

Four lessons stood out through delivery:

- Clear specifications produce reliable, generated implementation.
- Explicit governance constraints prevent insecure shortcuts.
- Ambiguous requirements produce ambiguous outcomes and rework.
- Specifications cannot anticipate every empirical property of the data being measured. Tuning numbers like a noise floor or a profile divergence tolerance belong in code, informed by the first production runs. SDD makes post deployment tuning safer because the surrounding contract is already fixed.

In production systems, senior engineers add the most value on architecture, constraints, and acceptance of quality, not on code drafting.





©2026 Nagarro

# Scope limits and non-goals

This system is designed for single VM sequential test execution one JMeter plan runs one execution host results are validated, and a structured report is delivered. It does not coordinate load across multiple agents or manage concurrent test slots.

It is not designed for:

- Real time distributed load generation requiring strict sub second coordination
- Environments without shared folder orchestration model
- High concurrency multi run scheduling
- Full cloud native queue abstraction

These are explicit design boundaries.

# Conclusion

This project shows that AI assisted development produces traceable, security conscious orchestration when governance is defined before implementation. The approach is repeatable for any team running structured performance testing at release cadence on VM based execution environments.


©2026 Nagarro

# About the author


## Sri Aditya

Senior Staff Engineer
(Performance Engineering)


Sri Aditya is a Senior Staff Engineer (Performance Engineering) with over 13 years of experience in performance engineering, architecture optimization, and performance testing. His expertise spans performance benchmarking, observability, cloud native platforms, and performance engineering for distributed systems. Passionate about Site Reliability Engineering (SRE) and AI driven automation, he focuses on applying AI to enhance performance engineering, streamline software delivery, and improve operational efficiency through practical, real world implementations.

# Reference:


**Spec Kit Repository:** https://github.com/github/speckit

**Spec Kit Tutorial**: https://github.github.com/speckit/quickstart.html

**Project Reference**: https://ghe.coxautoinc.com/SriErraguntla/SDDproject




©2026 Nagarro