Gatling

2025

Unlocking peak performance with load testing

The 10 load metrics every DevOps team should track in 2025

Why load testing still matters in a DevOps world

Atlassian and the State of DevOps Report popularized the **Four Critical DevOps Metrics or DORA metrics** (Deployment frequency, lead time for changes, change failure rate, and time to restore service). They are indispensable, but they often ignore what happens after code is deployed, when it faces real users at scale.

Load testing adds that missing diagnostic lens, turning velocity metrics into early-warning indicators rather than lagging statistics.

Continuous delivery has shrunk idea-to-prod cycles from months to minutes. Cloud elasticity promises "infinite" capacity at a keystroke.

Yet incidents such as retail flash-sale meltdowns and viral-launch outages prove that speed and scale are orthogonal. Even more insidious are the performance regressions that slip through with each new feature deployment: a recommendation engine that passes all tests but generates 10x more database queries, an API update that works perfectly in staging but creates cascading timeouts under real load, or a minor UI change that doubles page response times.

A system optimized for rapid change but blind to its saturation point is one deploy away from disaster. However, velocity comes with an invisible tax: performance regressions that slip through even the most rigorous functional testing.

Every new feature carries performance DNA that only reveals itself under real user load. The recommendation algorithm that works perfectly with test data becomes a database killer when processing millions of user preferences. Or the elegant new checkout flow that passes all UI tests doubles server response times when real shopping carts contain dozens of items.

The performance blind spot in modern DevOps

DevOps teams track deployment frequency, lead time, change failure rate, and mean time to recovery with religious precision. Nevertheless these velocity metrics are lagging indicators: **they tell you how often systems break but not why they fail or how to prevent the next outage.**

Industry analysis of 10,000+ production incidents reveals that 63% of all customer-impacting outages stem from performance degradations(1), not functional bugs. The gap is hiding in plain sight: **teams optimize for how fast they ship, not how their systems respond when real users arrive in force.**

THE COST OF PERFORMANCE BLIND SPOTS

- **User churn:** 40 % of users abandon a page that takes longer than three seconds to load.
- **Operational burn:** Unplanned performance firefighting can consume up to 20 % of engineering capacity.
- **Reputational damage:** Public post-mortems erode market confidence and brand trust.

Remember: Velocity ≠ resilience

Load testing metrics that matter

Modern systems demand specialized performance signals that expose nuanced bottlenecks across your entire stack. The following 10 core metrics provide comprehensive visibility into system behavior under load, exposing real bottlenecks, instability, and failure conditions that only emerge during high-traffic scenarios:

#1 | Error ratio (HTTP 5xx / unexpected codes)

Error ratio measures the percentage of requests your application attempts to handle but fails to complete successfully. This metric serves as your canary in the coal mine for deployment quality, revealing bugs, misconfigured feature flags, or broken connections to downstream services immediately after code changes go live.

Unlike functional tests that validate success paths, error ratio reflects how the system behaves when stressed, including how it fails. Teams should slice error data by endpoint and deployment version to quickly identify which specific code changes introduced problems. A sudden spike in error ratio, especially after a deployment, typically indicates application-level issues rather than infrastructure problems.

#2 | TCP connect-timeout rate

TCP connect-timeout rate tracks the percentage of network connections that fail to complete the standard three-way handshake within your specified timeout period. This metric is crucial because it reveals network-level saturation and load balancer capacity issues that won't appear in application logs.

When TCP timeouts increase while server CPU usage remains low, the bottleneck exists in your network infrastructure, not your application code. This client-side visibility is invaluable for diagnosing problems that server-side monitoring completely misses, such as firewall misconfigurations, network packet loss, or overwhelmed load balancers that drop incoming connections.

These are critical client-side metrics that reveal infrastructure issues your APM won’t catch.

#3 | TLS handshake-timeout rate

TLS handshake-timeout rate measures the percentage of SSL/TLS negotiations that either fail or exceed acceptable time thresholds. This metric becomes critical as organizations increase HTTPS adoption and implement stronger encryption standards. Rising TLS timeouts often indicate cipher suite mismatches, expired certificate chains, or CPU-intensive cryptographic operations overwhelming your servers.

The metric becomes particularly important during high-concurrency testing, where the computational overhead of establishing many simultaneous encrypted connections can create unexpected bottlenecks. Teams should also monitor TLS renegotiation patterns, especially with HTTP/2 implementations that can trigger additional handshake overhead.

#4 | Average response time

Average response time provides the mean latency across all requests during your load test period. While this metric offers useful trend analysis over days or weeks to detect gradual performance regressions, it should never be used for service level objectives or alerting thresholds.

Averages inherently hide the tail latency that actually impacts user experience. A system with 99% of requests completing in 50ms and 1% taking 10 seconds will show a misleadingly good average of 150ms. Use average response time exclusively for identifying slow drift patterns in system performance, but always pair it with percentile-based metrics for accurate performance assessment and operational decision-making.

#5 | Response-time standard deviation

Response-time standard deviation quantifies the variability in your system's latency, revealing performance inconsistency that impacts user experience predictability. High standard deviation indicates system jitter and instability, often caused by garbage collection pauses, resource contention, or inconsistent database query performance.

This metric becomes particularly valuable when standard deviation increases dramatically while mean response time remains stable, suggesting that a subset of requests experiences severe delays while the majority performs normally. Teams should investigate spikes in standard deviation alongside percentile metrics to identify whether performance problems affect specific user cohorts, geographic regions, or request types.

#6 | Response-time percentiles (P95/P99)

Response-time percentiles reveal the latency experienced by your slowest users, providing the most accurate picture of real-world performance impact. P95 represents the experience of your frustrated users, while P99 captures the performance that drives angry social media complaints and customer service calls.

Unlike averages, percentiles expose the tail latency that directly correlates with user abandonment and revenue loss. When P99 response times double while mean response time stays flat, you've identified a performance regression affecting a small but vocal user segment. Ensure your load testing tools use mathematically accurate percentile calculations, preferably HDR histograms, to avoid measurement errors that can hide critical performance issues.

#7 | Complete business-process duration

Complete business-process duration measures end-to-end transaction time for critical user journeys like account registration, product purchase, or document upload. This metric captures the true customer friction that individual API response times miss, revealing how multiple service calls, database queries, and third-party integrations compound to create the actual user experience.

A checkout process might show excellent individual API performance while the complete purchase journey suffers from cumulative delays across multiple steps. Teams should script entire business workflows during load testing rather than focusing solely on individual endpoints, ensuring performance optimization efforts target the user journeys that most directly impact business outcomes.

#8 | CPU usage (load generators)

CPU usage tracks how much processing power is consumed by your load generators during a test. It ensures that your test infrastructure is capable of generating the intended load without becoming a bottleneck itself.

If load generators approach 100% CPU utilization, they may fail to maintain target user injection rates or introduce artificial latency, skewing test results. Monitoring this metric helps validate the integrity of your test and guarantees that any observed performance issues originate from the system under test, not from a bottleneck in the test infrastructure itself.

#10 | TCP connection details

TCP connection details encompass the health of socket pools, connection reuse patterns, and network-level failures including connection resets and timeouts. This metric reveals connection pool exhaustion, keep-alive configuration problems, and network attacks like SYN floods that can overwhelm application infrastructure.

When active connection counts flat-line at specific thresholds, you've likely hit connection pool limits rather than application capacity constraints. Teams should track connection metrics per source IP to identify potential distributed denial-of-service attacks and monitor connection reset (RST) patterns that indicate network instability or firewall interference with long-running connections.

#9 | Heap memory usage (load generators)

Heap memory usage tracks how much memory is consumed by your load generators during a test. Sustained high usage can lead to excessive garbage collection, thread blocking, and inaccurate simulation behavior — especially under long or high-concurrency scenarios.

If heap usage on a load generator steadily increases or spikes above 85%, it may indicate internal inefficiencies (e.g., excessive object allocation or retained data). This can introduce artificial latency, cause injection rate drops, or even crash the generator mid-test. Monitoring this metric ensures the test infrastructure remains stable and does not skew performance results, especially when simulating large user loads or complex test scenarios.

For application-side heap usage, this metric must be tracked via your APM or observability platform.

METRICS	WHAT IT MEASURES	WHAT IT TELLS YOU	WHEN TO WORRY	TIPS FOR SUCCESS
Error ratio	Share of requests the app tried but failed to fulfill	Bugs, bad feature flags, broken downstream services	Spike after deployment	Slice by endpoint + version to find the guilty commit
TCP connect-timeout rate	Percentage of sockets that never finish the 3-way handshake	Load-balancer or network saturation	Rises while server CPU is low	Only visible from the client side
TLS handshake-timeout rate	Percentage of SSL sessions that exceed threshold or fail	Cipher mismatch, expired cert, CPU-heavy handshakes	Climb at high concurrency	Track renegotiations on HTTP/2
Average response time	Mean latency across all requests	Slow drift over days indicates creeping regression	Use for trend analysis only	Never use for SLOs—hides tail latency
Response-time standard deviation	Latency variability and consistency	Jitter and system instability	Standard deviation balloons while mean stays flat	Pair with P95/P99 to locate bad cohorts

METRICS	WHAT IT MEASURES	WHAT IT TELLS YOU	WHEN TO WORRY	TIPS FOR SUCCESS
Response-time percentiles	Tail latency that real users actually experience	Small slice of very slow requests	P99 doubles but mean remains OK	Verify your tool's percentile math accuracy
Complete business-process duration	End-to-end journey time for critical workflows	True customer friction across entire user flows	Slower checkout but fast individual APIs	Script the whole flow, not just endpoints
CPU usage	CPU utilization of load generators during a test	Whether the test infrastructure can sustain the desired load without introducing bottlenecks or artificial latency	Sustained CPU usage above 80%, especially when paired with rising response time percentiles (e.g., P95)	Always validate that load generators aren't CPU-saturated before attributing slowdowns to the system under test
Heap memory usage	JVM heap memory consumed by load generators during a test	Detect memory pressure, risking GC pauses, thread blocking, or generator instability	Heap usage steadily increases or spikes above 85%, especially in long or high-concurrency tests	Monitor to prevent injection slowdowns or mid-test crashes
TCP connection details	Health of socket pools and keep-alive connections	Exhausted pools and connection reset storms	Open-connection count flat-lines at pool size	Track per source IP to catch SYN floods

Load testing is your competitive advantage

Organizations no longer see load testing as merely preventing outages.

Modern DevOps requires both velocity and resilience. By implementing these 10 metrics with Gatling Enterprise, your organization transforms load testing from a reactive bottleneck into a competitive advantage that enables faster, more confident, and more profitable software delivery.

It's now a competitive differentiator enabling:

- **Accelerated testing cycles:** TRAY reduced load testing time from 3 days of manual work to hours of automated analysis, while TUI implemented automated CI/CD pipeline integration for continuous performance validation

- **Infrastructure optimization:** JioStar demonstrated efficient scaling capacity, supporting 30M concurrent users during Indian Premier League matches and scaling from 15M to 30M users within 90 seconds during critical live events

- **Superior user experience:** Performance improvements directly impact user satisfaction - TUI achieved 50% mean performance improvement with response times improving from 1.5s to 0.7s, while TRAY reduced response times by up to 90% with 200% improvement in API response performance

- **Engineering efficiency:** Teams report significant time savings in performance validation - TRAY's engineering team can now complete performance data analysis in hours rather than the previous 3-day manual process, enabling faster development cycles

Gatling

gatling.io

Gatling is the leading solution for modern load testing, enabling developers and organizations to deliver fast, reliable applications at scale. With its powerful open-source and enterprise platforms, Gatling empowers teams to test APIs, microservices, and web apps in real-world conditions. Trusted by thousands of companies worldwide, Gatling is the performance backbone for development, QA, and DevOps teams building the next generation of software.