Gatling

2026

WHITEPAPER

Case study: performance testing across the software industry

How high-growth software teams turn reliability into a competitive advantage

The business case is simple: reliability is retention

In a subscription economy, your software's performance is your renewal rate. In a developer-tools market, it is your reputation. In a cloud-native architecture, a single degraded service is a customer complaint three hops away.

The organizations that understand this do not treat performance testing as a pre-launch checklist. They treat it as a continuous signal — one that tells them, on every merge, in every pipeline, whether the software they are about to ship will behave under the conditions their customers will actually create.

This whitepaper maps the four moments where performance risk concentrates in software organizations, shows how the teams that get this right build it into their delivery process, and demonstrates how Gatling Enterprise makes continuous performance intelligence achievable at the speed modern software requires.

Four moments. Four different failure patterns

Software performance risk does not distribute evenly across the development lifecycle. It concentrates at specific points — moments where the assumptions baked into your architecture meet the reality of what your users actually do.

Four moments define the performance risk landscape for software companies today:

1. **Product launch and scaling events:** when traffic spikes are known, anticipated, and make-or-break

2. **CI/CD and release pipelines:** when speed of delivery creates invisible regression risk

3. **API-first and microservices architectures:** when distributed systems create emergent failure modes at integration points

4. **Legacy modernization and cloud migration:** when lifting and shifting breaks performance contracts that were never written down

Each moment has a distinct risk profile, a distinct failure pattern, and a distinct testing strategy. And each moment carries more risk than it used to, because software ships faster and recovery windows are shorter.

01 | PRODUCT LAUNCH & SCALING EVENTS

When traffic spikes make it or break it

In software, you don't know what production load looks like until you're in it.

SaaS companies with strong retention grow 1.8x faster than their peers. The launch moment is where that retention is won or lost.

THE RISK JOURNEY OF A PRODUCT LAUNCH

One launch event. Multiple systems in motion.
One window to get it right.

1. **Feature complete**
   Development and QA sign off on the release.
   Risk: Performance was never part of the definition of done.
2. **Staging validation**
   Testing runs in pre-production environments.
   Risk: Concurrency patterns don't reflect what happens when real users arrive simultaneously.
3. **Go-live**
   The launch event begins.
   Risk: Real user traffic doesn't behave like test scripts.
4. **Post-launch ramp**
   Traffic builds as adoption grows.
   Risk: The system survived launch. It degrades under sustained growth.
5. **Partner and integration traffic**
   Enterprise clients go live. Integration partners connect. **Risk:** Third-party and partner traffic patterns add unexpected pressure to services.
6. **Customer support signal**
   Users report degradation. Monitoring alerts fire.
   Risk: Performance issues surface through support tickets and churn.

STATS AT A GLANCE

$4.5M per hour That's the peak season downtime cost for large online retailers, when inadequate load testing is the primary root cause

1.8x faster growth rate for SaaS companies with Net Revenue Retention above 100%, driven by reliability that earns and keeps customer trust

The risk: During Black Friday 2023, Harvey Norman and J.Crew's platforms suffered a five-hour outage, costing roughly $775,000. Both incidents traced back to inadequate load handling.

Test beyond the launch event

Validate autoscaling behavior
Test how your infrastructure scales under realistic ramp patterns, not just peak load. Confirm autoscaling thresholds trigger at the right time and don't overshoot under burst conditions.

Simulate client onboarding
Model the traffic pattern of multiple large clients going live simultaneously. Don't let a new enterprise account be your first real load test.

Test the user journey under load
Benchmark individual endpoints, but validate the end-to-end user flow — session creation, core operations, and downstream service calls.

Reduce risk with Gatling.
Welcome to Continuous Performance Intelligence.

AI capabilities
Automated run summaries surface what went wrong without manual analysis. IDE integration and MCP server support bring load testing into the workflows your engineers already use.

SLOs and compliance scoring
Define response time and error rate targets directly in Gatling Enterprise Edition. Every run returns a compliance score, not a pass/fail, a precise percentage. Know exactly how long your system held the line.

Test as code
Write load test scenarios in Java, JavaScript or Scala. Versioned, reviewed, and living in the same repository as your application. Plug directly into your CI/CD pipeline.

Private locations
Load generation runs inside your network perimeter. No traffic leaves, no firewall exceptions. Test what's actually in production.

When speed of delivery creates invisible regression risk

The largest software failure in history was caused by a software update that bypassed adequate testing.

Every release pipeline carries that risk at some scale.

THE RISK JOURNEY OF A SINGLE RELEASE

One commit. From development to production.
Six points where performance risk accumulates invisibly.

1. **Code commit**
   A dev pushes a change written or accelerated with AI.
   Risk: Larger changesets introduce more unpredictable performance behavior under real load.
2. **Build and unit tests**
   Automated tests run against the commit.
   Risk: Unit tests validate logic, not load behavior. Performance regressions introduced here are invisible
3. **Integration tests**
   Services are tested together in a shared environment.
   Risk: Inter-service performance is typically untested at this stage.
4. **Load test — if it exists**
   Performance testing is executed, if it's in the pipeline.
   Risk: When it is, it often runs manually outside the pipeline with thresholds that don't match the SLOs.
5. **Staging deployment**
   The release candidate reaches a pre-production env.
   Risk: Staging is often under-scaled with synthetic traffic that doesn't reflect production load shapes.
6. **Production release**
   The release ships to users.
   Risk: Performance degradation is discovered by monitoring after users are already affected.

STATS AT A GLANCE

47% of testers do not consider load testing part of their scope

7.2% increase in delivery instability coming from AI adoption

The risk: 68% of the organizations conduct performance testing, and 55% of them encounter difficulties due to the unavailability of test environments. This indicates that over half of the organizations may deploy software into production without proper testing and mainly performance testing to assess the system reliability under real-world conditions.

Test beyond functional correctness

Regression gates in every pipeline
Embed load tests as CI/CD gates with SLO-based pass/fail criteria. A p95 regression that fails staging never reaches production.

Baseline comparison across releases
Track performance across every release. Know whether each deployment improves, holds, or degrades under load before it ships to users.

Validate AI code under load
When AI tools accelerate commit velocity, test frequency must match. Validate larger changesets before they compound into incidents.

Reduce risk with Gatling.
Welcome to Continuous Performance Intelligence.

AI capabilities
Automated run summaries surface what went wrong without manual analysis. IDE integration and MCP server support bring load testing into the workflows your engineers already use.

SLOs and compliance scoring
Define response time and error rate targets directly in Gatling Enterprise Edition. Every run returns a compliance score, not a pass/fail, a precise percentage. Know exactly how long your system held the line.

Test as code
Write load test scenarios in Java, JavaScript or Scala. Versioned, reviewed, and living in the same repository as your application. Plug directly into your CI/CD pipeline.

Private locations
Load generation runs inside your network perimeter. No traffic leaves, no firewall exceptions. Test what's actually in production.

When distributed systems create emergent failures at integration points

74% of organizations are now API-first.
Only 45% of API teams employ performance testing

The number that has declined every year for four years, while AI-driven API traffic surged 73%.

THE RISK JOURNEY OF AN API REQUEST

One request. Multiple services in motion.
Multiple ways to fail silently.

1. **Client request**
   A user action reaches the API.
   Risk: Authentication and session validation under load creates latency that compounds downstream.
2. **API gateway**
   The request routes to the appropriate service.
   Risk: Connection pool exhaustion and timeout aren't tested until they become the bottleneck.
3. **Business logic service**
   Core processing executes.
   Risk: Under concurrent load from multiple callers response time degrades in ways that weren't modeled.
4. **External dependency**
   A third-party API, data provider, or service is called.
   Risk: A 200ms increase in a dependency you don't own becomes your SLO breach, not theirs.
5. **Database and cache layer**
   The query runs.
   Risk: Cache invalidation behavior under load was never tested.
6. **Async processing and queues**
   The response returns. Background processing begins.
   Risk: Sustained traffic builds queue backlogs that are invisible in synchronous response time metrics

STATS AT A GLANCE

57% of developers cite API performance as the top reliability risk

45% of cloud-native orgs report cascading failure as primary incident type

The risk: 62% of companies now generate revenue directly from APIs. For 21%, API monetization represents more than 75% of total revenue. An API-first architecture distributes financial risk across every service boundary.

Test beyond peak policy events

Distributed load across services
Test each microservice under concurrent load from all upstream callers, not in isolation. Identify which service degrades first and under what combination of traffic.

Third-party dependency simulation
Model realistic failure modes for external APIs and data services. Test how your system behaves when they slow down — not only when they're healthy.

Backpressure and queue behavior
Validate how your system handles sustained load at the boundary between synchronous and asynchronous processing.

Reduce risk with Gatling.
Welcome to Continuous Performance Intelligence.

AI capabilities
Automated run summaries surface what went wrong without manual analysis. IDE integration and MCP server support bring load testing into the workflows your engineers already use.

SLOs and Compliance scoring
Define response time and error rate targets directly in Gatling Enterprise Edition. Every run returns a compliance score, not a pass/fail, a precise percentage. Know exactly how long your system held the line.

Test as code
Write load test scenarios in Java, JavaScript or Scala. Versioned, reviewed, and living in the same repository as your application. Plug directly into your CI/CD pipeline.

Private locations
Load generation runs inside your network perimeter. No traffic leaves, no firewall exceptions. Test what's actually in production.

When lifting and shifting breaks unwritten performance contracts

A system in transition. Old and new running together.

Every handoff is a point where untested load behavior can surface.

THE RISK JOURNEY OF A MIGRATION

One launch event. Multiple systems in motion.
One window to get it right.

1. **Legacy core**
   The existing system continues to process requests.
   Risk: Stable under known load patterns, but brittle when a new integration layer introduces real-time demand
2. **Modernization layer**
   Legacy code, APIs, and microservices share space
   Risk: Concurrency patterns don't reflect what happens when real users arrive simultaneously.
3. **Cloud-native services**
   New services run alongside legacy infrastructure
   Risk: They scale differently, fail differently, and respond to load patterns in different ways.
4. **Data synchronization**
   Records migrate and sync between old and new systems.
   Risk: Migration pipeline performance under production-volume data was never tested
5. **Parallel operation**
   Systems handle traffic simultaneously during transition.
   Risk: The combined load on legacy infrastructure during parallel operation.
6. **Migration milestone**
   A release, cutover, or compliance checkpoint approaches.
   Risk: Untested behavior under production load surfaces at the moment the business has committed to a date.

STATS AT A GLANCE

70% of modernization projects experience significant performance degradation

40% higher post-deployment failure rate with inadequate testing

The risk: When legacy maintenance already consumes up to 80% of the IT budget, there is no margin to absorb a failure that was foreseeable and testable. The organizations that recover fastest are the ones that validated their new architecture under load before the cutover window, not after.

Test beyond everyday transaction volume

Load test simultaneously

During parallel operation, validate that both legacy and modernized components handle concurrent load without degrading each other.

Validate cutover scenarios

Simulate the traffic pattern at the moment of migration cutover. Confirm the new system handles production-volume concurrency from day one.

Test with production data

Migration performance often diverges at scale. Validate query and processing behavior under realistic data volumes before the cutover, not after.

Reduce risk with Gatling.
Welcome to Continuous Performance Intelligence.

AI capabilities

Automated run summaries surface what went wrong without manual analysis. IDE integration and MCP server support bring load testing into the workflows your engineers already use.

SLOs and compliance scoring

Define response time and error rate targets directly in Gatling Enterprise Edition. Every run returns a compliance score, not a pass/fail, a precise percentage. Know exactly how long your system held the line.

Test as code

Write load test scenarios in Java, JavaScript or Scala. Versioned, reviewed, and living in the same repository as your application. Plug directly into your CI/CD pipeline.

Private locations

Load generation runs inside your network perimeter. No traffic leaves, no firewall exceptions. Test what's actually in production.

From test data to business intelligence

Engineering teams generate performance data. What most organizations lack is the infrastructure to turn that data into decisions.

Test runs produce reports that live in dashboards no one checks, get attached to tickets that close without resolution, or trigger alerts that the on-call engineer silences at 2 a.m. because they do not have enough context to act.

The problem is not the volume of data. It is the absence of interpretation.

The organizations that close this gap share three practices:

1. They define SLOs before they write tests: A performance test without a service level objective is a measurement looking for a threshold. SLOs define what "good" means in terms the business understands — the response time that keeps users engaged, the error rate that keeps support queues manageable, the throughput that meets contractual commitments to enterprise customers. When SLOs are defined first, every test result is immediately interpretable as pass or fail. No expert analysis required. No context needed at 2 a.m.

2. They treat performance regressions like functional bugs: In organizations with mature performance intelligence, a regression in CI fails the build the same way a failing unit test does. It gets a ticket. It gets assigned. It gets fixed before the next release. This is a cultural commitment — performance as a first-class quality attribute, not a post-launch investigation.

3. They connect test results to production observability: The gap between load test behavior and production behavior is the gap between what you tested and what actually happened. Closing it means connecting Gatling test environments to the same APM and observability platforms used in production — so test results and production metrics are on the same timeline, with the same instrumentation. What passed in testing should hold in production.
When it does not, you need to know immediately and know why.

Gatling Enterprise Edition is built for software companies

PRIVATE LOCATIONS

Deploy load generators inside your VPC, your Kubernetes cluster, or your on-premises data center. Test traffic never leaves your network.

SLOs

Define p95 and error rate thresholds in Gatling. Breach them and the pipeline stops automatically.

ENTERPRISE SECURITY

Plugs into the governance frameworks your organization already enforces.via SSO (OIDC/SAML), role-based access controls, and audit logs.

AI ASSISTANT

Accelerates how developers create, explain, and optimize, and understand performance tests and Gatling simulations

CI/CD INTEGRATION

Trigger runs, enforce thresholds, and gate promotions from GitLab CI, GitHub Actions, or any pipeline.

TEST AS CODE

Write scenarios in Java, Kotlin, JS/TS, or Scala using the same toolchain your backend and version control engineers already use.

SUPPORTED PROTOCOLS

HTTP, WebSocket, SSE, gRPC, JMS, and MQTT support APIs, streams, messaging, microservices.

COMPARISON AND TRENDS

Run comparison and trends help teams spot regressions, track performance over time, and see whether each release improves, holds, or degrades under load.

INFRASTRUCTURE AS CODE

Spin injector fleets up and down as IaC. Nodes appear in your VPC, disappear after the run.

OBSERVABILITY

Integrations with APM tools enable you to correlate load test behavior with infrastructure metrics

What software companies actually test with Gatling?

Aircall: catching regressions before customers do

Aircall is a cloud-based phone system and call center software platform used by more than 17,000 businesses worldwide.

As a real-time communications platform, Aircall's performance requirements are unforgiving: calls cannot buffer, queue metrics cannot lag, and customer support integrations cannot time out. For Aircall's customers — who are themselves running customer-facing operations — Aircall downtime is directly visible to their own customers.

Gatling shifted how Aircall manages performance risk: from reactive incident response to proactive detection in pre-production. Performance regressions that would previously have been discovered by customers were caught in the pipeline.

✓ **THE RESULT?** Product and engineering gained visibility into the system's performance posture across every release cycle, not only during incidents.

Attentive is the AI-powered SMS and email marketing platform used by more than 8,000 brands, including some of the most recognized names in retail and consumer products. The platform sends billions of messages annually, with peak send events — Black Friday, Cyber Monday, major retail campaigns — concentrating enormous throughput into short windows.

Gatling's distributed load generation architecture allows Attentive to scale test load to production-representative volumes without maintaining a parallel infrastructure of equivalent scale. Test as Code means the load tests that validate Black Friday readiness are version-controlled, peer-reviewed, and evolved alongside the platform capabilities they test.

✓ **THE RESULT?** Attentive goes into Black Friday knowing — not hoping — that the infrastructure will hold.

They share the same performance reality: they are B2B SaaS platforms where their customers' business outcomes run directly on top of their infrastructure. For both companies, a performance failure is an immediately visible, attributable event that puts customer contracts at risk. That's why the cost of getting it wrong is measured not in incident tickets, but in churn.

SOPHOS skello attentive CIRCLE K

Adobe sopra steria korem

A performance risk framework for software companies

Use these questions to assess where your organization stands; bring them to your next leadership meeting as a starting point for the conversation.

1 COVERAGE: ARE WE TESTING WHAT ACTUALLY MATTERS?

Core user journeys, public APIs, authentication flows, and data pipelines deserve different testing cadences than internal dashboards and admin tools. Most organizations test what's easy to reach, not what's critical to protect.

- [ ] **Can you name your five most business-critical user journeys?**
- [ ] **Were each of them tested under realistic peak conditions in the last 90 days?**
- [ ] **Do you test your API layer, authentication flows, and background processing separately?**
- [ ] **Are test cadences tied to business criticality, not just release schedules?**

2 REALITY: DO OUR TESTS REFLECT WHAT ACTUALLY HAPPENS?

Smooth ramps and clean datasets produce clean results and false confidence. Production doesn't behave like a staging environment; launch spikes, viral traffic events, and retry storms are foreseeable patterns that test environments routinely fail to simulate.

- [ ] **Does your test environment include the same authentication layers and rate limiters as production?**
- [ ] **Do your load scenarios account for third-party dependencies?**
- [ ] **Do you simulate foreseeable spike events: product launches, partner integrations, seasonal peaks?**
- [ ] **Do load generators run inside your own infrastructure, behind your VPN or VPC?**

3 INTERPRETATION: DO WE KNOW WHAT TO DO WITH THE RESULTS?

A test without a decision path is just a report. Every performance test should answer a specific question, with a named owner and a clear path for what happens if it passes or fails.

- [ ] **Does each recurring test have a named owner or team accountable for the results?**
- [ ] **Is there a specific, documented question each test is designed to answer?**
- [ ] **Is there a predefined decision path for both pass and fail outcomes?**
- [ ] **Did your last performance test produce a ship/hold decision, not just a dashboard?**

4 OWNERSHIP: WHO IS ACCOUNTABLE?

Performance can't live with one team. Developers, platform engineers, and engineering leaders each need a defined role; business leadership needs visibility into trends over time.

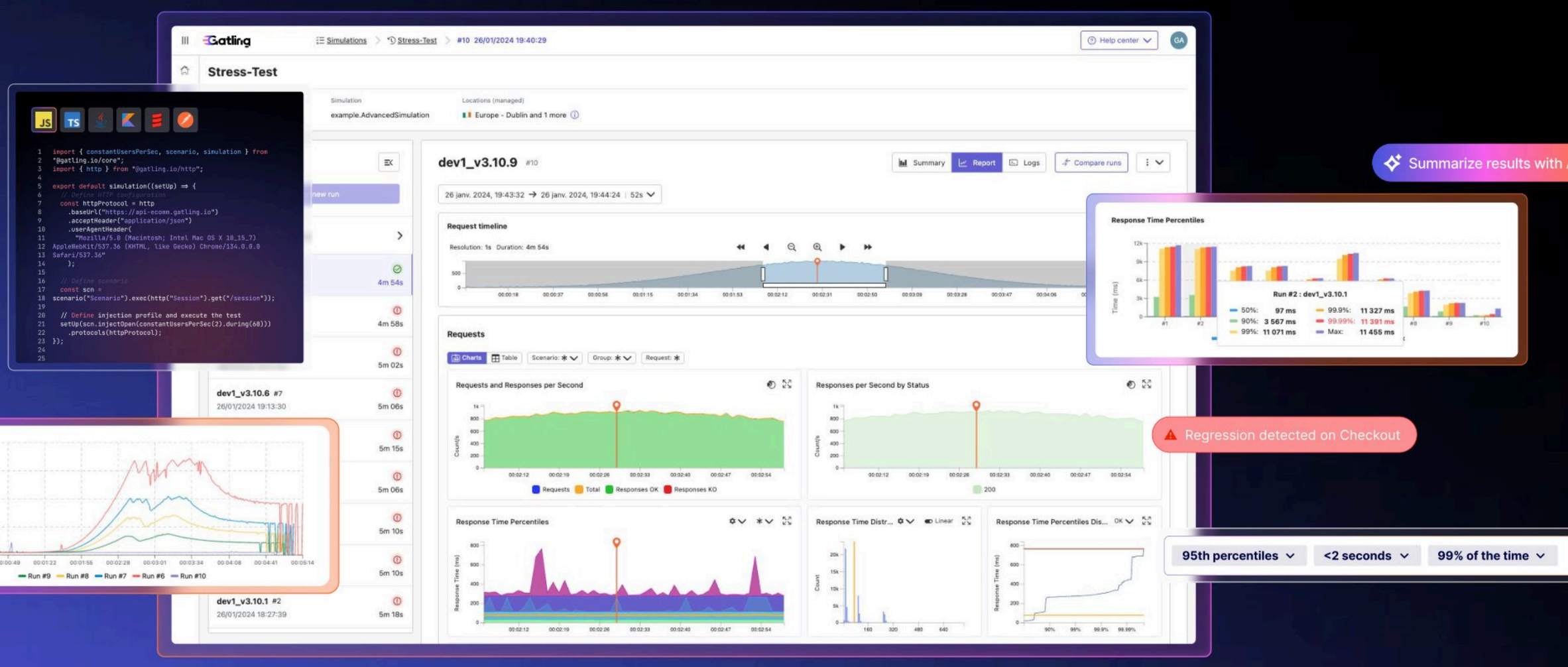
- [ ] **Do developers own performance for the code they ship?**
- [ ] **Does business leadership have regular visibility into performance trends?**
- [ ] **If a critical API degraded 40% today, is it clear who makes the ship/hold call?**
- [ ] **Do your load tests simulate realistic user journeys and transaction sequences?**

Reliability is now a competitive advantage

In a market where alternatives are always one search away and subscription churn is the primary business risk, the organizations that win are the ones whose software keeps working when it matters most. Functionally correct, yes, but also performant, resilient, and reliable at the scale of real usage

The four delivery moments mapped here — product launches, CI/CD pipelines, API-first architectures, and legacy modernization — are where that reliability advantage is built or lost. They are the moments when assumptions about system behavior meet the reality of production load. And they are the moments when continuous performance intelligence is the difference between catching a problem in a pipeline and discovering it in a customer complaint.

Gatling Enterprise Edition is the platform built for this reality. Test as Code makes performance tests a first-class engineering artifact. CI/CD integration makes performance gates a first-class delivery control. Private locations, distributed load generation, and protocol coverage make it possible to test the architectures modern software actually uses — not the simplified approximations that fit inside a single load injector SLO compliance scoring, AI assistance, and observability integration turn test data into the business intelligence that drives decisions.



Gatling

Gatling Enterprise Edition is the platform purpose-built to deliver Continuous Performance Intelligence: transforming load testing from a technical activity into an organizational capability that business leaders can govern, product teams can engage with, and engineering teams can 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.

Whether you're scaling APIs, migrating to the cloud, or handling flash traffic spikes, Gatling helps you deliver fast, reliable performance.

Ready to impment continous performance intelligence?

See how AI-assisted performance testing
can take you to the next level

Talk to an expert ›