Gatling

2025

EBOOK

Build or buy your load testing solution

for high-scale, engineering-driven organizations

Why teams build (and why it's getting harder)

Spinning up a single test is simple. Running a trustworthy, distributed, pipeline-driven performance platform that serves multiple teams is not. Many engineering teams begin with open source to validate behavior. But over time, maintaining orchestration, analytics, integrations, and security turns into a hidden burden.

Users already expect your application to feel as fast as the best in the world. That raises performance testing from a late-stage validation step to a continuous quality engine. Modern architectures—microservices, edge, hybrid cloud—make real test scenarios distributed by design. And raw metrics aren't enough; teams need actionable insights, gated pipelines, and reliable governance.

When evaluating options, most teams compare:

- Open source solutions — low cost, fast ramp, hands-on
- Enterprise / managed platforms — built for scale, insight, and team adoption

The risks and costs of building

Many teams start building for control, flexibility, or cost. But the deeper complexity often emerges later:

- **Distributed orchestration**. You become responsible for coordinating generators across locations, retry logic, time sync, and result collection,plus handling partial failures gracefully. This is ongoing engineering, not a one-off script.
- **Metrics firehose & analysis.** High-cardinality time-series data must be aggregated and retained in a way that lets teams do run trends and run-to-run comparisons to spot regressions quickly. Without this, people fall back to manual diffing and intuition.
- **Integration drift.** CI vendors, runners, and APM tools evolve. Keeping pipelines green and alerts useful means maintaining plugins, CLIs, and webhooks continually. Every broken connector is lost confidence in testing.
- **Security & governance.** SSO, RBAC, group-to-role mapping, quotas, auditing, and shareable reports are mandatory for cross-team adoption. Building and operating them well is non-trivial.
- **Opportunity cost.** Every hour spent keeping a homegrown platform alive is an hour not spent shipping product. Unless performance testing is your product, this is rarely the highest-leverage use of your engineers.

> DIY can be great for exploring. At scale and over time, the operational surface area—distribution, analysis, integrations, and security—becomes the work.

Validating your approach

Use an open-source solution or lighter tooling to validate your key flows before investing in scale. Define your critical user journeys, test them in staging or dev, and verify that assertions make sense for your use cases. Gatling supports multiple languages (JavaScript/TypeScript, Java, Scala, Kotlin) and even imports Postman collections to help you cover real flows faster. Begin with a couple of key scenarios, simple assertions, and a first CI job to confirm connectivity and basic performance.

You'll know your stack is stretching when you need to:

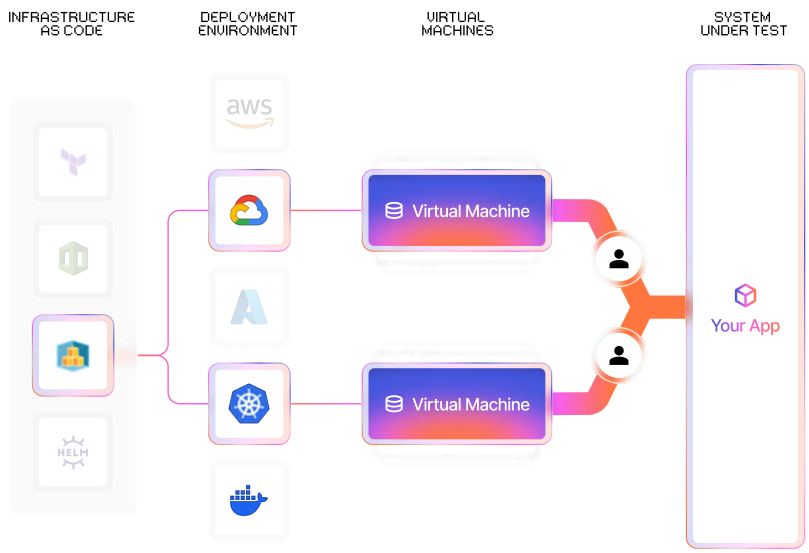
- Run large and multi-region tests reliably
- Compare runs and analyze regression trends without manual effort
- Support different roles (dev, QA, SRE, management) from a shared platform
- Automate tests in your CI pipeline with stop criteria/gates
- Enforce governance via SSO, RBAC, quotas, and safe result sharing

That's when a more robust platform becomes essential.

Scaling your tests horizontally

Vertical scaling (one big host) hits limits quickly and can skew results. Distributed generation lets you:

- Model global traffic patterns and regional failover scenarios
- Avoid single-host bottlenecks and noisy neighbors
- Keep sensitive traffic in-network (e.g., via Private Locations) while still using a central UI and APIs for control and analysis.



Tools like Gatling let you automate the provisioning of multiple load generators (in cloud or in-network).

> This is where "build" turns into "operate a product." A platform that already handles generator lifecycle, placement, and network constraints will save significant toil.

Embed performance into your delivery pipeline

Performance testing should protect releases, not slow them down. Wire tests into CI so they run on PRs, merges, and on a schedule for long-running scenarios. Use stop criteria/performance gates to fail the pipeline when SLOs are breached, and rely on run comparisons to see exactly what changed across builds. That's how teams keep velocity while raising quality.

Collaborating across teams

Successful programs make performance a shared responsibility. Your solution needs to work for:

- Developers: code-first authoring, fast feedback loops
- QA/SRE: dashboards, alerts, drill-downs, trend analysis
- Leads/Managers: shareable reports, project-level views, budgets/quotas
- Security/IT: SSO, RBAC, audit trails, and tenancy boundaries

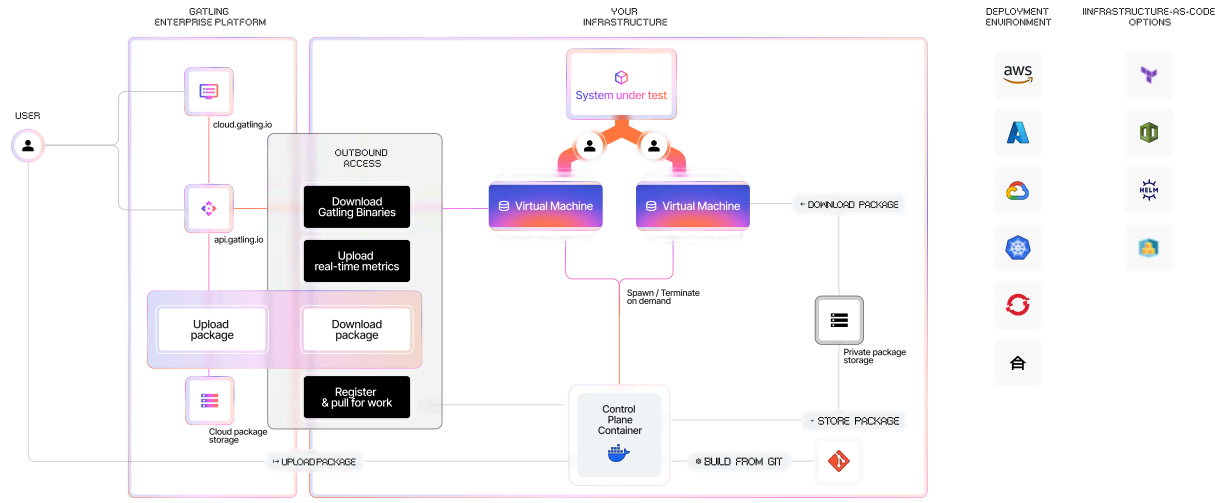
> If adoption requires manual workarounds or permission silos, performance testing becomes an afterthought instead of baked into engineering culture.

Making your solution reliable (beyond "it runs")

At scale, a platform must deliver:

- Control-plane availability and auto-recovery when parts fail
- Data aggregation and retention without runaway storage costs
- Redundant capacity & regional flexibility so tests still succeed when one path fails
- Deployment choices (cloud, private, hybrid) to meet compliance, security, and data-sovereignty constraints

> A mature platform ships these as table stakes; a homegrown stack must evolve them deliberately.



Layers of orchestration and control in a mature load testing platform

Assessing Total Cost of Ownership (TCO)

Don't compare "license vs. free." Compare lifecycle costs and time to value:

- **Engineering time:** build + continuous upkeep of orchestrators, data, dashboards, plugins
- **Infrastructure:** compute, storage, egress, and capacity planning
- **Integration drift:** CI, APM, and identity systems change
- **On-call:** who gets paged when the test platform flakes?
- **Opportunity cost:** features and fixes your team didn't ship because they were platform-sitting

> For many orgs, a managed platform ends up more economical and far less risky once you factor in continuity, governance, and organizational adoption.

Build vs. Buy — Pros & Cons

Dimension	Build In-house solution	Buy Enterprise platform
Time to value	Fast to prototype or run initial POCs, but slows dramatically as you scale across teams or regions.	Ready to use from day one, with faster onboarding and immediate value at scale.
Distributed orchestration	You design and operate control, scheduling, retries, and data aggregation.	Orchestration handled by the vendor, with consistent APIs and lifecycle management.
Metrics & analysis	Stand up Time-series database + retention + trends/compare yourself; maintain indefinitely.	Strong data aggregation. Trends and Comparison to catch regressions quickly.
Automations & CI/CD	Maintain scripts/plugins as ecosystems evolve.	Native plugins and "any CI" path; stop criteria and gates to protect releases.
APM/alerts	Write and maintain integrations per tool.	Off-the-shelf hooks/webhooks documented and supported.
Security & governance	Build SSO/RBAC/quotas; keep mappings and policies current.	Enterprise SSO, RBAC, quotas, and audit built-in.
Deploy anywhere	Custom work to keep traffic in-network/on-prem with central control.	Private Locations: run generators inside your cloud or on-prem with one-way control.
TCO & team focus	Lower upfront cost; rising op-ex and on-call; ongoing roadmap burden.	License cost offset by less platform toil and a predictable evolution path.
You should build when	Highly unique, strategic requirements tied to your product.	
You should buy when		Common, repeatable needs at scale; extensible via APIs, and integrations.

Should we keep building?

A practical checklist

- [ ] **We run distributed, multi-region tests with confidence.**
  Generators are orchestrated reliably (scheduling, retries, aggregation) and don't require heroics to keep green.
- [ ] **We have owners for dashboards, trends, comparisons, and data retention.**
  Trends & comparison exist and are used in every regression review.
- [ ] **Our CI/CD hooks are stable across tools, and easy to keep current.**
  We can trigger load tests from our main CI(s) and enforce gates/stop criteria without custom glue.
- [ ] **APM/alerting integrations don't drift.**
  Results and thresholds flow to the observability stack via supported, documented paths.
- [ ] **SSO/RBAC/quotas and auditable sharing are in place.**
  Identity, roles, group mapping, and usage controls are first-class features, not ad-hoc scripts.
- [ ] **Performance checks are pipeline-gated.**
  Pipelines fail automatically on defined SLO breaches; teams don't "ship and hope."
- [ ] **We handle high-scale load predictably.**
  We can push very large user volumes with consistent reliability and cost control (not just "one big box").
- [ ] **Results are easy to share and consume.**
  Stakeholders can open interactive reports, compare runs, and export data without bespoke tooling.
- [ ] **Onboarding new users is low-friction.**
  Inviting users, assigning roles, and getting them productive is a documented, UI-driven flow.
- [ ] **We can generate load in-network and scale globally when needed.**
  Traffic to private or internal endpoints remains fully within our network, while we also support seamlessly invoking public locations for high-scale, cross-region injection in the same test run.

> It isn't black-or-white—and it isn't a forced path. If several boxes are unmet now, buying directly is often the highest-leverage move. You can still use Gatling Community Edition locally for authoring (e.g., code in JS/TS/Java/Scala/Kotlin or import Postman) while the platform handles scale, analysis, and governance.

If load testing matters, your tool should too.

Your users demand speed and reliability, your load testing platform must deliver the same, with power, precision, and scalability. **Gatling Enterprise Edition** offers a complete platform built for modern teams, distributed systems, and real-world performance demands, aligned along five product pillars:

Analyze smarter & act faster

Gain real-time visibility with dashboards, trend comparisons, and actionable insights.

Create tests your way

Build tests via code, low-code, or no-code, import Postman, script in JS/TS or Java, or design visually.

Unlock automations

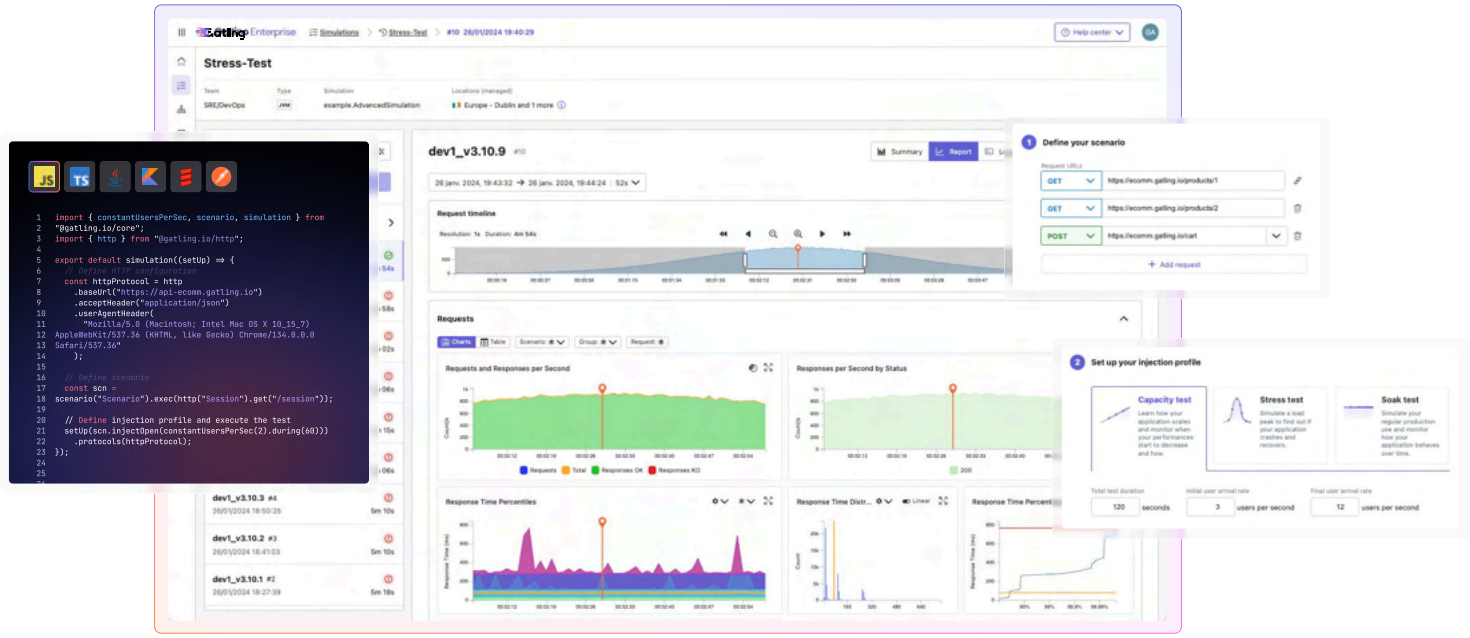
Trigger simulations via CI/CD or API, apply stop criteria, and gate releases with performance thresholds.

Collaborate & share results easily

Use RBAC, SSO, quotas, and shared reports. Share results via Slack, Teams, or Jira.

Deploy load generators anywhere

Run tests from Gatling managed regions, your cloud, or on-prem.



Gatling

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.

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

Ready to evaluate Enterprise Edition?

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

Talk to an expert ›