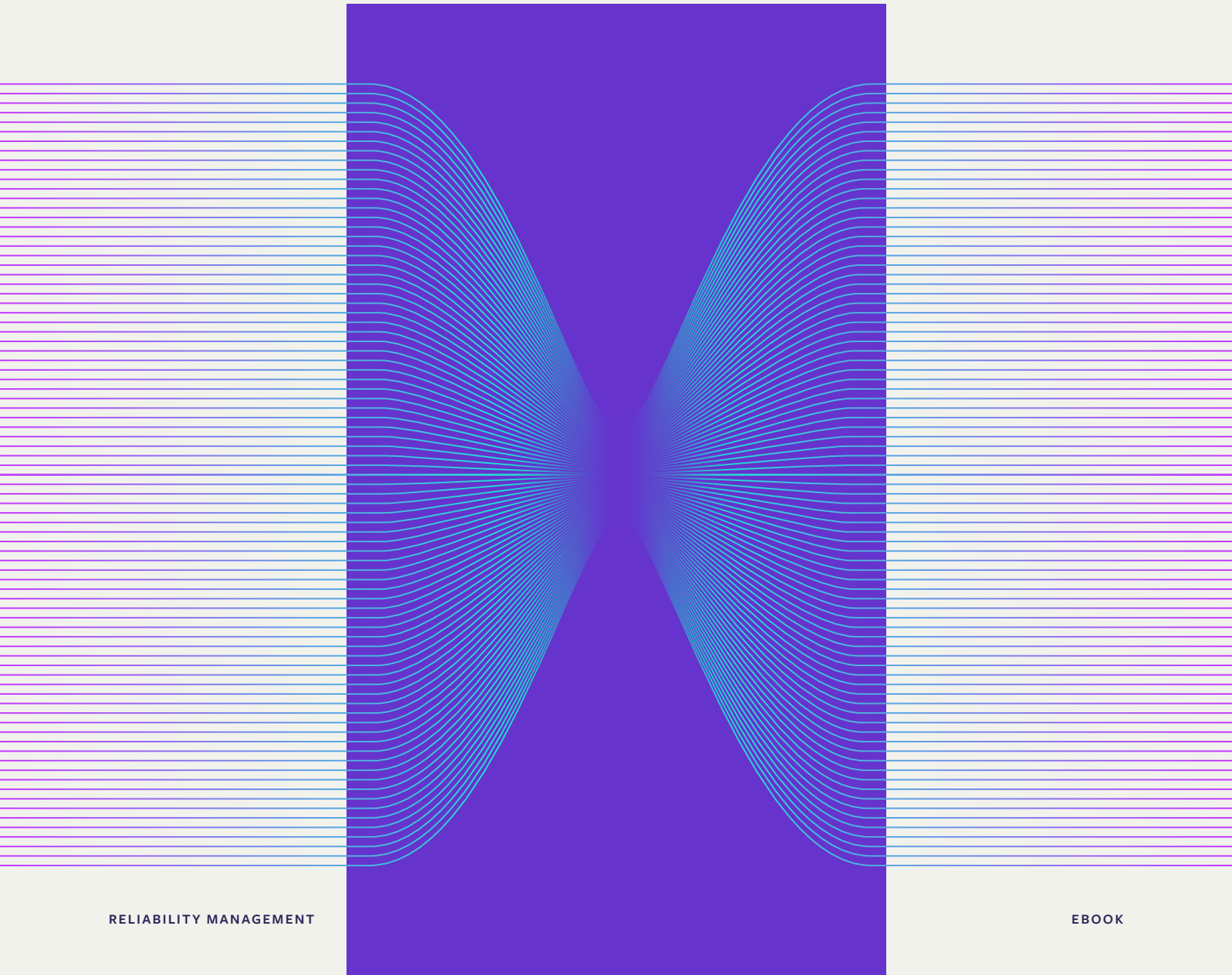
Gremlin

Closing The Reliability Gap

HOW TO USE TESTING, SCORING, & AUTOMATION TO PROACTIVELY MANAGE AND IMPROVE RELIABILITY



RELIABILITY MANAGEMENT

EBOOK

TABLE OF CONTENTS

Facing the Gap

If you're running complex, distributed systems, you're likely facing the reliability gap.

At its worst, the gap shows up as brand- and revenue-impacting incidents and outages. But it's often less obvious and just as expensive: reliability risks can slow software development, force your best engineers to fix fundamental infrastructure issues, and overwhelm organizations with manual, time consuming processes.

Regardless of the symptoms, the issue is the same: there's a gap between where your reliability is and where we know we want to be.

Reliability is more important than ever. Customers expect the services they use will always be available. If customers try to use your service only to find a poor experience or down altogether, they're more likely to move to a competitor.

And yet, **reliability is harder than ever to get right.** With the rise of DevOps, microservices, and distributed infrastructure, applications are growing larger and more complex. Applications are changing more rapidly, with many teams pushing changes to production several times a day. Our mental models haven't kept up with these systems, and it's less clear what good reliability actually looks like. We often can't baseline our systems, prioritize remediations, and track progress.

The problem is **there hasn't been a standardized way to measure and improve reliability that fits modern development.** Some practices have emerged, such as hiring SREs, adopting observability and incident response tools, and running Chaos Engineering experiments. These each play a role but are collectively insufficient to close the reliability gap at scale.

Where common reliability approaches fall short

OBSERVABILITY & INCIDENT RESPONSE	Reactive by nature; doesn't prevent issues not yet seen
CHAOS ENGINEERING	Not measurable; hard to adopt and scale
SRES	Expensive and difficult to scale; inconsistent standards and practices
SLOS & SLIS	Measure *un*reliability; doesn't indicate where to improve

Clearly, we need a new approach.

Gremlin has been working with companies to build world class reliability programs since 2016. Prior, our founders were pioneering the practice at companies like Amazon and Netflix. We've seen industry standards emerge for what reliable systems look like and learned first hand alongside our customers. With Gremlin Reliability Management, we've tackled and distilled the most difficult aspects of reliability into something teams can use to drive real progress.

Let's get started.

Setting a Standard

Gremlin's Reliability Management Platform includes three elements that have traditionally been some of the most complex and difficult aspects of reliability: scoring, testing, and automation. These elements are essential to build and support a broader reliability program.

SCORING: USE A STANDARDS-BASED APPROACH

Gremlin uses Reliability Scores to provide a standardized measure of reliability. Scores provide a clear indication of how well teams are meeting their reliability objectives and can be used throughout the organization to systematically manage and improve reliability.

WHY USE A RELIABILITY SCORE?

1 – Consistent, standardized measure of reliability across services, teams, and infrastructure.

2 – Proactive indicator of reliability strengths and risks that doesn't rely on incidents or on-call heroes.

3 – Proves resiliency against common causes of failure with clear pass/fail indicators

Using a score

For IT executives and centralized reliability functions: scores provide a quick view of the organization's overall reliability posture, including recent changes, strengths, and risks. If scores are low–especially for critical services–leaders know where to direct their teams to focus on improving reliability.

For application and service owners: scoring provides an ongoing level of confidence–when paired with regular, automated testing–that the service is meeting reliability standards of the organization. If there is a drop in the service's score due, teams can quickly find and address the cause to maintain reliability.

For SRE and DevOps teams: scoring provides the ability to set clear expectations of what service reliability looks like. Teams can prove they are making their service more reliable, without waiting for incidents to reflect in backward-facing metrics. Paired with an automated test suite to generate the score, these teams can improve reliability test coverage across the organization to free up time for more valuable work.

Designing and generating a score

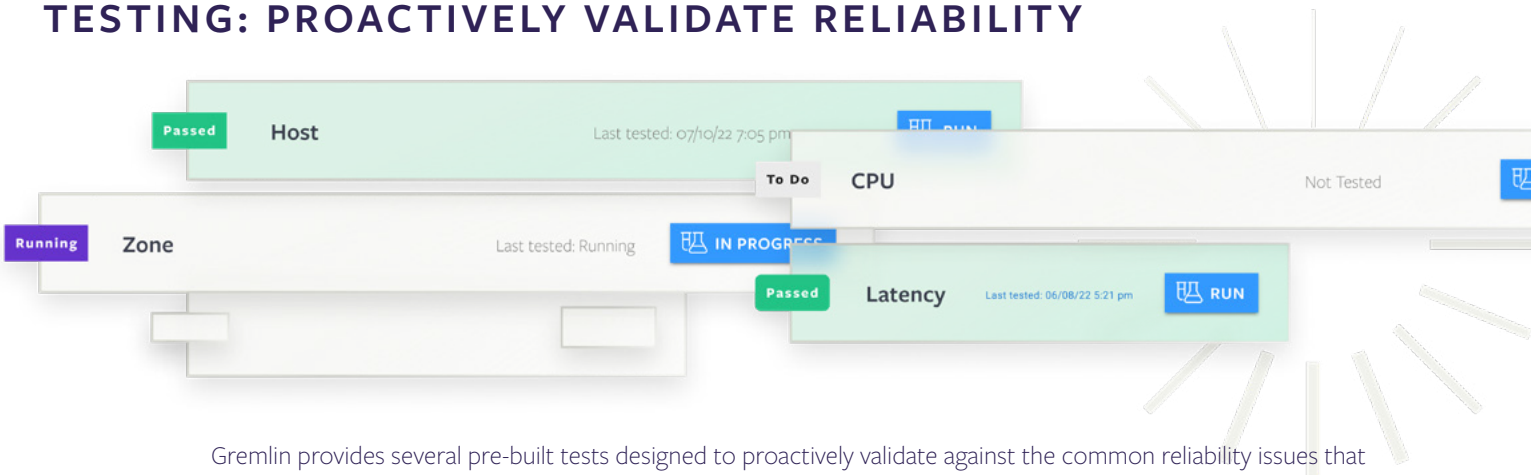
How do you generate a score for reliability when there are so many variables? Reliability isn't random: there are industry best practices and years of learning about the real-world causes of failure to lean on.

A score should start with the most important attributes of modern distributed systems: scalability, redundancy, and dependency reliability. For each attribute, define common failure scenarios and how you expect your systems or services to respond to those scenarios. For example:

- **Scalability:** if I reach my system's limit CPU, memory, or disk space, does my system automatically scale and add capacity?
- **Redundancy:** if one of my hosts, availability zones, or regions fails, can I automatically failover and keep my service running?
- **Dependency reliability:** if I lose access to a key dependency due to network latency or a network outage, can customers still use my service?

By basing a score on these attributes, weighted by category, you're covering the essential elements of reliability. But how do you validate that your systems meet these requirements?

TESTING: PROACTIVELY VALIDATE RELIABILITY



Gremlin provides several pre-built tests designed to proactively validate against the common reliability issues that generate a score. Each test runs against a service and contributes points to the final score. These points vary depending on whether your service remains healthy during the test. The individual test scores are compiled into a final score representing how well you can trust your service to remain available.

Where to test

Modern software environments are complex and distributed with countless moving parts, and testing could be done at many layers. However, reliability testing is best approached on a per-service basis. Services (or microservices), are independently deployable units of functionality that provide a single function within a broader application, like a checkout service or authentication service. Gremlin uses services as the main unit of reliability measurement and improvement.

Team may start running reliability tests in a non-production environment; however, the ultimate goal is to run and pass tests in production. Because tests are designed to simulate real-world scenarios, it's best to validate that services can withstand these issues in real-world environments. Gremlin includes a number of safeguards to make testing in production safe and secure.

How to determine pass or fail

To determine whether a service passes or fails a reliability test, Gremlin looks at its Golden Signals through monitoring and observability tools: latency, traffic, error rate, and resource saturation. These metrics represent the four most important attributes of a system from your end user's perspective.

If any of the four golden signals become unhealthy during a test, this indicates the service isn't resilient to that test. Gremlin automatically stops the test, rolls back to the previous state, and marks the test as a fail. If the signals remain healthy during the duration of the test, you've validated your systems are resilient to that failure mode and the test is marked as a pass.

Remediation after failing a test

Failing a test provides insight into where your systems are vulnerable and identifies action items to improve reliability. For example, if a host redundancy test brings your service offline, then that's a clear indication to improve that service's redundancy. You can address this by enabling autoscaling, then repeat the test to ensure your solution works as expected.

Gremlin uses services as the main unit of reliability measurement and improvement.

AUTOMATION: ENSURE RELIABILITY ON A CONTINUOUS BASIS

Systems change over time: new infrastructure gets provisioned, code changes get deployed, and emergent behaviors can arise. These changes can cause regressions; the same reliability tests that passed one week might fail the next week. This is why we need to test regularly.

Ideally, we'd run a full suite of tests for every change to production, but this can be time-consuming. As a balance, Gremlin recommends testing at least once per week and will automatically flag test results over a week old as "expired." This is to remind teams to run tests regularly or to use the built-in auto-scheduling service to automate testing at least weekly.

Teams can also automate reliability practices into their software delivery lifecycle by pulling scores into their existing tools. For example, we can use the Gremlin API to retrieve a service's reliability score during the CI/CD process. If the score drops below a certain threshold, we can trigger our CI/CD tool to block new builds until the score increases or the service owner reviews the cause of the low score. This way, we can catch and revert unreliable code before it can impact our customers.

THE PATH TO RELIABILITY AUTOMATION

1 – **Baseline** Determine reliability posture of each service with standardized reliability testing

2 – **Remediate** Identify, prioritize and implement reliability improvements.

3 – **Automate** Maintain standards with continuous testing and scoring

Conclusion

Proactive scoring, testing, and automation are some of the most difficult elements of reliability, but are essential for teams to have a clear path towards making their systems more reliable. With the right processes and tools, it's possible to close the reliability gap while accelerating software delivery.

To learn more, get a demo at **gremlin.com/demo.**

Gremlin