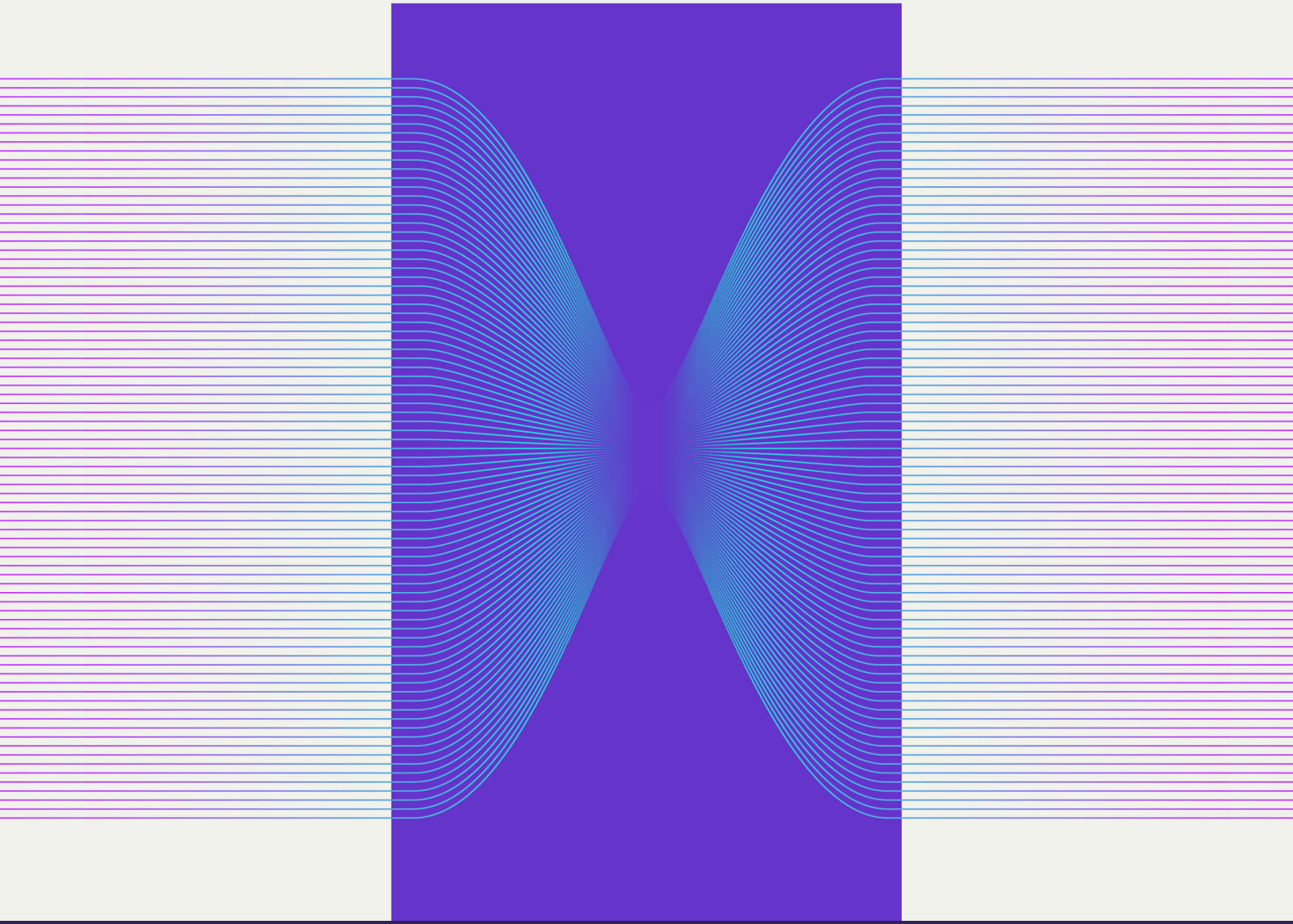
Gremlin

Closing the Reliability Gap

BUILDING A CULTURE OF RELIABILITY THROUGH TESTING, SCORING, AND AUTOMATION



RELIABILITY MANAGEMENT

EBOOK

Table of contents

3 **Introduction**

5 **Test for reliability risks**

6 Scan configurations to detect common risks

7 Test for specific failure modes with reliability tests

9 Understand systems with custom Chaos Engineering experiments

10 **Align around Reliability Scores**

11 Why use a Reliability Score?

12 How the score is built

13 **Continue and automate testing**

15 **Conclusion: Bridging the reliability gap demands collaboration**

16 **Further Reading**

Introduction

Reliability is more important than ever. Customers expect the services they use to always be available.

If customers try to use your service only to find a poor experience or that the service is down altogether, they're more likely to move to a competitor. And if your customer is a business, then an outage or incident can cause real damage, something they're definitely going to remember when it comes time to renew their contract.

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.

That is the reliability gap—the separation between where your reliability is and where you know you want it to be.

At its worst, the gap shows up as brand- and revenue-impacting incidents and outages. But it can also show up as reliability risks that are less obvious and just as expensive as they slow software development, force your best engineers to fix fundamental infrastructure issues (instead of launching new features), and overwhelm organizations with manual, time consuming processes.

The problem is that there hasn't been a standardized way to measure and improve reliability that fits modern development. Some practices have emerged, such as site reliability engineering, 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 - Can't predict what hasn't already been seen

Incident Response - Reactive by definition; can't prevent unplanned incidents

SLOs & SLIs - Measure unreliability; don't indicate where to improve

Chaos Engineering - Good for exploration and specific risks, but hard to scale

Site Reliability Engineering - Expensive and difficult to scale; dependent on other teams

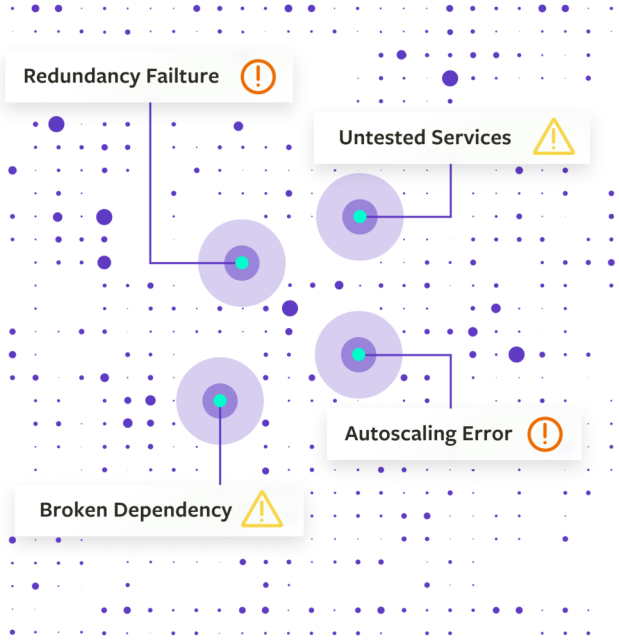
QA and Performance Testing - Lacks visibility into real-world, less-than-ideal conditions

If you're going to close the reliability gap, you need to be able to see the current reliability of your systems, prioritize remediations to fix the reliability risks most likely to impact customers, and track reliability progress over time.

When you do this, you won't be forced to wait until something breaks to find out if there's a problem. By testing ahead of time and standardizing around core reliability metrics, you'll be able find and fix reliability risks before they become incidents.

Test for reliability risks

Reliability risks are potential points of failure within your system.



They're similar to the idea of security vulnerabilities. Having security vulnerabilities doesn't mean that your system is automatically compromised, but they are a weak point where a breach could happen, so the vulnerability should be addressed before someone exploits it.

The same is true with reliability risks. Your system may be running fine right now, but these are places where your system could break. And you can prevent an incident by locating and remediating the cause of the risk before it causes an outage.

Let's look at resource scalability as an example. The service may run just fine with a normal amount of traffic, but a sudden surge may cause its memory or CPU usage to spike. And if it's not set up correctly, that spike could bring the service, and everything dependent on it, down.

Unfortunately, our modern, complex architectures often have far more reliability risks than we'd like. Working closely with our customers, **we've found critical reliability risks** present in nearly every organization, such as lack of zone redundancy and misconfigured autoscaling. For example, our data shows 26% of Kubernetes deployments had zero zone redundancy and 80% failed to utilize more than two zones (a best practice recommended in the AWS Well-Architected Framework). These misconfigurations can have serious impacts on system reliability. In this case, if one zone is unavailable, which is common, a full quarter of deployments would be offline.

A combination of scanning for common risks, reliability testing, and custom Chaos Engineering experiments can help you locate reliability risks. Let's look at how you can use this combination using Gremlin, a SaaS solution that helps enterprise organizations find and fix their reliability risks before they turn into incidents.

Scan configurations to detect common risks

Like with addressing security vulnerabilities, the first step is to scan for any common or known risks.

Gremlin's Detected Risks helps you find and fix the most common causes of infrastructure outages without running Chaos Engineering experiments or reliability tests

	Total Risks	Kubernetes Memory Request	Availability Zone Redundancy	Kubernetes Memory Limit	Kubernetes Liveness Probe	Application Version Uniformity	Kubernetes CPU Request
datadog-cluster-agent	4/6	✗	✗	✗	✓	✓	✗
gremlin-control-plane-agent-segment	0/6	✓	✓	✓	✓	✓	✓
gremlin-alfi-api	0/6	✓	✓	✓	✓	✓	✓
gremlin-control-plane-apl-segment	0/6	✓	✓	✓	✓	✓	✓
datadog-agent	3/6	✗	✓	✗	✓	✓	✗
gremlin-event-worker	0/6	✓	✓	✓	✓	✓	✓
gremlin-support-api	0/6	✓	✓	✓	✓	✓	✓

Detected Risks are high-priority reliability concerns that Gremlin automatically identified in any environment where the Gremlin agent is installed. These risks can include misconfigurations, bad default values, or reliability anti-patterns. Gremlin prioritizes these risks based on severity and impact for each of your services. This gives you near-instantaneous feedback on risks and action items to improve the reliability and stability of your services.

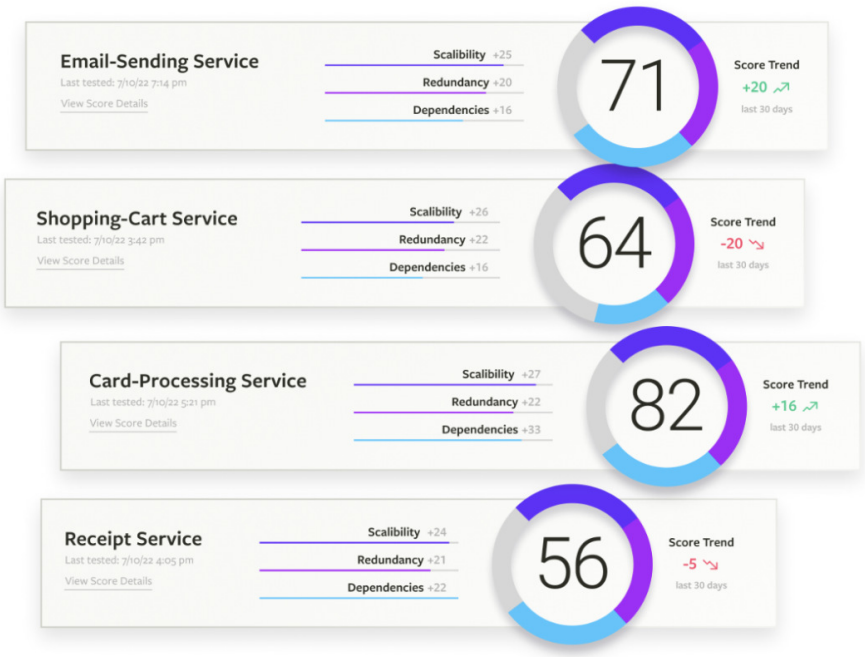
These risks can include:

- CPU Requests
- Liveness Probes
- Availability Zone Redundancy
- And more...
- Memory Requests
- Memory Limits
- Application Version Uniformity

The goal with Detected Risks is to quickly give you actionable data that you can use to uncover and remediate the most common reliability risks.

Test for specific failure modes with reliability tests

Gremlin provides several pre-built tests designed to proactively validate against common reliability issues.



These tests use Fault Injection to stress systems while watching the monitors you've set in your observability tool. If your systems can run the test without triggering an alert, you pass. If not, tests are halted and marked as a fail—there is work to do to remediate this risk.

These include tests under three broad categories:

Scalability

Tests if your service behaves as expected when system resources (such as CPU or memory) are limited or exhausted.

Redundancy

Tests whether your service is reliable when one of your hosts, zones, or regions is unavailable.

Dependencies

Dependencies are any independently maintained software component used by a service to provide features or functionality. Dependencies can be hard (required) or soft (not required), and this test checks whether your service performs as expected when a dependency is slow, unavailable, or has an expiring security certificate—a surprisingly common cause of incidents.

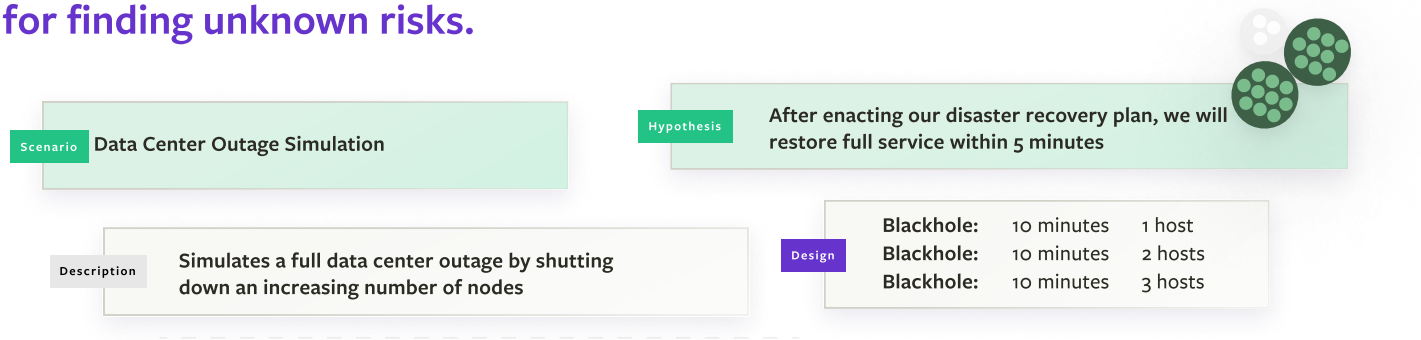
While Gremlin includes comprehensive tests out of the box, many organizations modify these tests or add their own to set the standard for reliability consistently across the organization or meet external compliance requirements. For example, Gremlin's Zone Redundancy test runs for five minutes, but many banks need to provide evidence to regulators that they can withstand the loss of a zone for three hours. Gremlin makes it simple to adjust the parameters and test for this.

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.

When you first start testing, you may want to run reliability tests in a non-production environment that closely mirrors production, but 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.

Understand systems with custom Chaos Engineering experiments

Chaos Engineering experiments are incredibly effective for finding unknown risks.



The practice of Chaos Engineering originated with Netflix in the late 2000s. At the time, Netflix was transitioning from local servers to the cloud, which is, by its nature, much more ephemeral.
To help train their engineers, they introduced Chaos Monkey, which would turn off and replace servers randomly in an effort to replicate the "chaos" of the cloud.

As a practice, Chaos Engineering has come a long way since those humble roots. Using technology like Fault Injection, Chaos Engineering is still an incredibly effective practice for finding specific risks, validating resilience to certain failure modes, and understanding how your application behaves under less-than-ideal conditions.

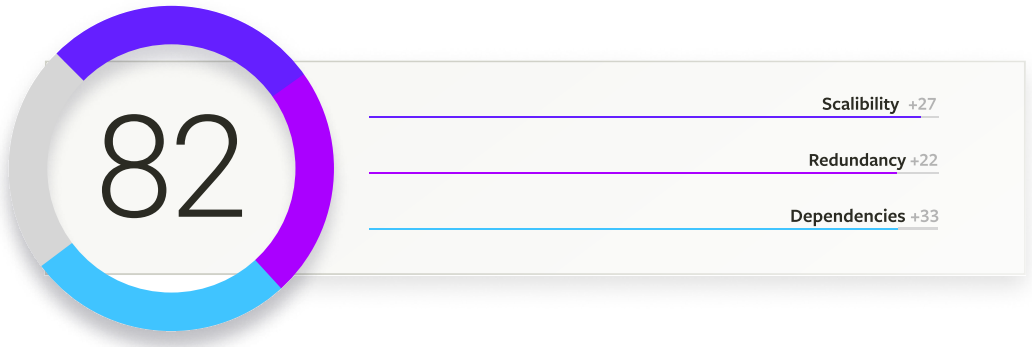
Between Detected Risks scanning for the most common risks and reliability tests verifying services against the most common failure modes, the majority of potential failures are covered. But every system and architecture is unique, and Chaos Engineering fills in the rest of the gaps so you can confidently test your entire system.

With Gremlin, you can build custom experiments using a wide variety of targeting and experiments types, then save them as custom scenarios that you can add to your automated testing set up.
Designed to run anywhere your services are, it can be used to test failure modes on anything from on-prem to cloud to kubernetes to serverless.

Between those three, you'll be able to get a complete, accurate picture of the reliability posture of your systems. But then what do you do with it?

Align around Reliability Scores

Once you've found your reliability risks, the next step is remediation. Data is critical.



Competing priorities and high-velocity organizations can make it hard to get the time and resources needed to remediate the issue. And if you do get the time to fix the issue, it can be hard to prove the value of your efforts by pointing to the damaging outage that didn't happen.

This is where reliability metrics come in. Most organizations track reliability by using backward-facing metrics like downtime, MTTR, or the number of incidents and outages. All of these are important metrics, but they're also all negative metrics, where progress is built around decreasing them, or having less of a bad thing—which can only get you so far. When metrics improve, it begs the question: are we actually reducing risk, or just getting lucky?

In order to make consistent, scalable, repeatable improvements to reliability, you need metrics that show the current reliability posture of your systems—and can show when you've improved that reliability posture.

Gremlin uses Reliability Scores to give you that metric.

Reliability Scores are a standardized measure of reliability based on the results of Reliability Tests and specified custom scenarios. 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.

If you're just getting started with reliability testing, Gremlin provides a score based on a suite of common tests that are already defined for you. If you want to meet specific requirements for your organization or industry, or ensure compliance with external regulators, you can create a score based on custom scenarios as part of the test suite you define for your organization.

Why use a Reliability Score?

Reliability Scores give your team a clear metric for reliability to align around. They give your team:

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

With Reliability Scores, you change the reliability discussion with concrete metrics that everyone can align around to make informed decisions about how they prioritize risks.

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.

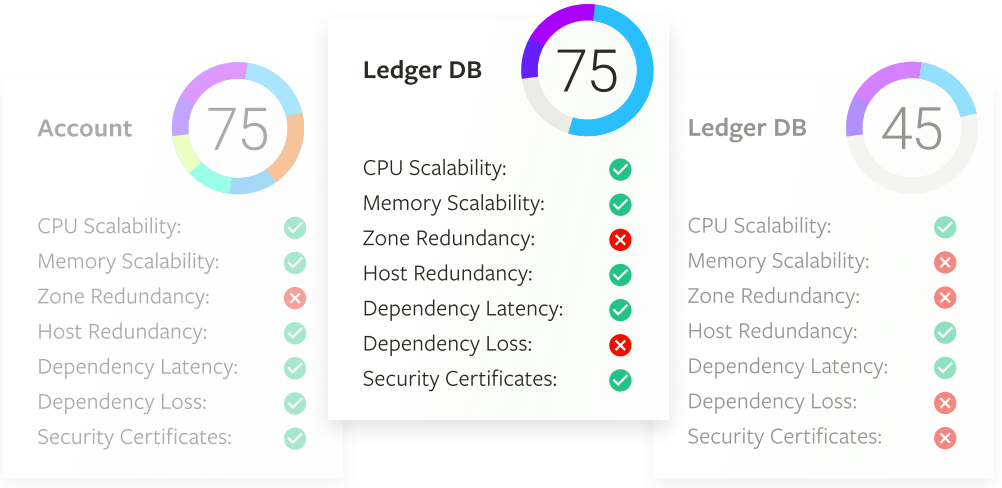
For application and service owners: scores provide an ongoing level of confidence that the service is meeting reliability standards of the organization. If there is a drop in the service's score due, teams can quickly find and address the cause to maintain reliability.

For SRE, DevOps, Performance, and QA teams: scores provide 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.

Say you have a critical shopping cart service that you always want to have a score of 90 or higher. And if it drops to 80, engineering cycles need to be devoted to keep it up and running. But at the same time, you might have an internal environment setup service that has a threshold of 50. It might be inconvenient for engineers to wait for an extra 10 minutes for an environment to get set up, but it isn't nearly as critical.

How the score is built

The score is based on the number of passed tests compared to the number of failed or unfinished tests.



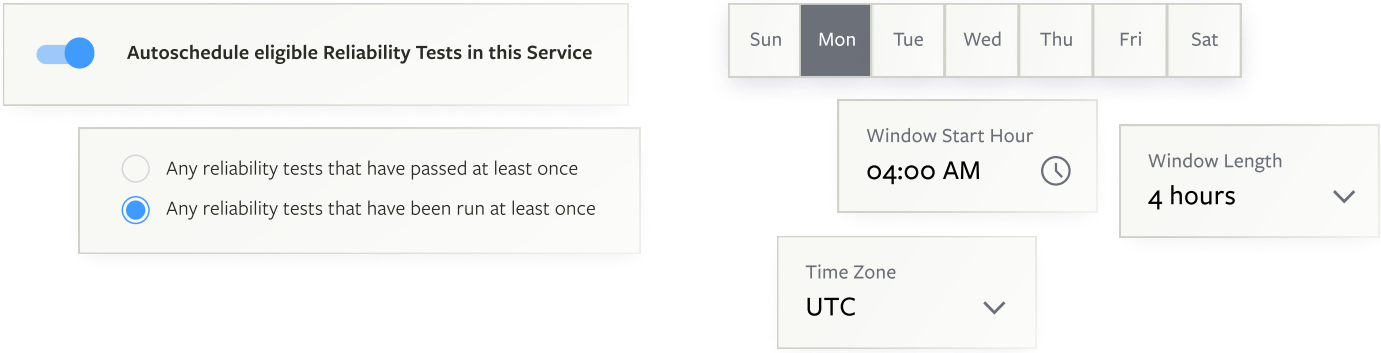
For the purposes of scoring, a test that wasn't performed (or hasn't been performed recently) counts the same as a failed test. This is based on the concept of, "If you can't measure it, then it didn't happen." When you don't know the results of a test, it's just as much of a reliability risk as a failed test because you have no idea whether your service will perform as expected.

To determine whether a service passes or fails a Reliability Test, Gremlin looks at its Health Checks by deeply integrating with your existing monitoring and observability tools. These metrics, such as latency, traffic, error rate, and resource saturation, are probably already used today, and are whatever you define as representative of system health from your end user's perspective.

If any check becomes 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.

Continue and automate testing

Systems change over time. New infrastructure gets provisioned, code changes get deployed, and a service that passed a reliability test could now fail it.



Testing regularly in an automated fashion is a path to continuously validate system reliability and get notified when things change.

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, you can use the Gremlin API to retrieve a service's test results and reliability score during the CI/CD process. This way, you can catch and revert unreliable code before it can impact our customers.

Automated testing also creates a crucial map of your reliability posture over time. By tracking the Reliability Scores for each service, you can judge the effectiveness of your reliability efforts, the impact of code changes (such as if a score drops from 80 to 50 after a deploy), and get an idea of the accrual rate of technical debt in the form of remediations that need to be addressed. Some organizations allow teams with a high enough score to be exempted from the manual quarterly or annual testing, since this work is already running in the background.

THE PATH TO RELIABILITY AUTOMATION

1. **Test -** Determine reliability posture of each service with reliability testing

2. **Report -** Use dashboards to share the results with leadership and service owners

3. **Remediate -** Identify, prioritize, and implement reliability improvements.

4. **Automate/Repeat -** Maintain standards with continuous testing and scoring

Conclusion: Bridging the reliability gap demands collaboration

When you get right down to it, reliability is the responsibility of everyone in the technology organization.

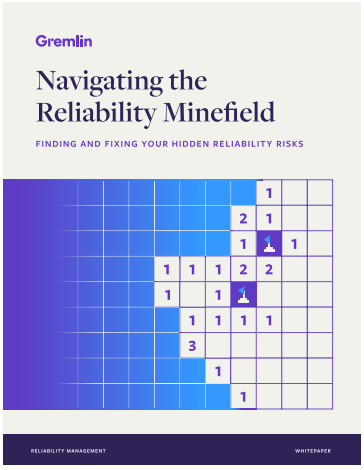
Technology leaders need to make it a priority, be able to see the current status, and track the results of their investment. Engineers and service owners need actionable data they can use to drive remediation efforts. And reliability owners (like Site Reliability Engineering teams) need to define methodology to measure, track, and align reliability efforts.

These are all created by the combination of reliability testing, Reliability Scores, and continued automation to create a visible system of record focused on making real reliability improvements

Because with the right shift of your approach, tools, and cultural mindset, it's possible to finally close the reliability gap.

Further reading:

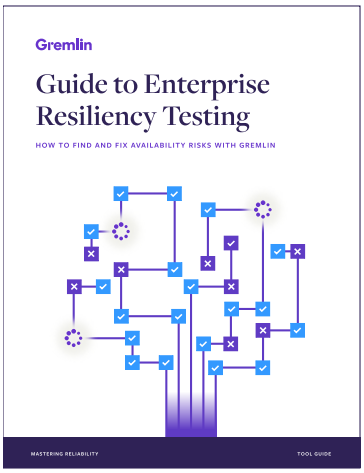
Ready to start closing the reliability gap? Check out these resources to help you take the next step on your reliability journey:



Navigating the Reliability Minefield:
Find and Fix Your Hidden Reliability Risks

Get the processes and tools to avoid outages before they happen with this whitepaper. Learn how to identify and track reliability risks, prioritize fixes, and prove your results to your organization. Includes a free spreadsheet template to help you build your own reliability tracker.

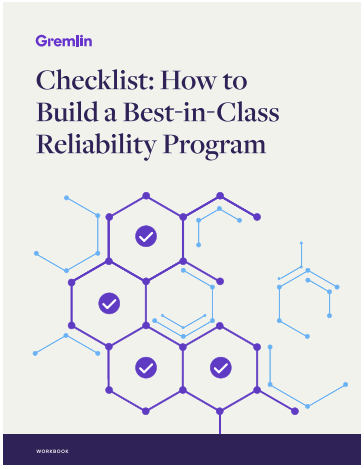
LEARN MORE



Guide to Enterprise Resiliency Testing:
How to Find and Fix Availability Risks with Gremlin

Reliability testing might look daunting, but it doesn't have to be.
This guide explains how to use Gremlin's comprehensive reliability experiments to solve real-world use cases. Improve your systems' fault tolerance, redundancy, scalability, and more with this whitepaper.

LEARN MORE



Checklist: How to Build a Best-in-Class Reliability Program

Based on work with reliability leaders at Fortune 100 companies, Gremlin identified four pillars and 18 traits common in successful reliability programs to align organizations, get crucial buy-in, and measurably improve their reliability.
Use this checklist to make sure your reliability program checks all the right boxes to make it successful.

LEARN MORE

Gremlin

Gremlin is the Enterprise Reliability Platform that helps teams proactively test their systems, build and enforce reliability and resiliency standards, and automate their reliability practices organization-wide.

Learn more at **gremlin.com**.