

# Ensure Application Performance Amid Chaos



## Introduction to Chaos Testing

Chaos testing is a subset of Chaos Engineering devoted to testing. Chaos Engineering is the discipline of experimenting with a system to build confidence in the system's capability to withstand turbulent conditions in production. By ensuring the system can withstand chaotic fluctuations in its state you can be confident in its ability to handle unexpected real-world issues. This could include things like infrastructure, network, or power failures at various points in the system.

It's hard to imagine a software development team that doesn't do any testing. Whether unit, integration, [functional](#), [performance](#), [security](#) or even manual—software testing is widely accepted as a best practice in the software

development lifecycle (SDLC). Usually, companies plan and create test exercises ahead of time. They often involve applying frequent test cases to expected events. However, the bugs and vulnerabilities that set the stage for major system failure, exploitation, or intrusion result from unexpected events.

The primary difference between ordinary testing and chaos testing is the scale and the results. Chaos testing tries to ensure that even in the event of chaos, our software systems keep functioning and watching client requests, even if entire parts of the system crash.

This article walks you through chaos testing, how it works, and why and how you should use it.

## Flash Point Paper



### Questions to Ask Yourself About Chaos Testing:

- *What is chaos testing?*
- *What is the difference between chaos testing and chaos engineering?*
- *What kind of challenges can be solved by chaos testing?*
- *What are some examples of real-world applications that use chaos engineering today?*
- *Can you simulate chaos attacks on your systems?*
- *Are there any tools available today that embrace chaos testing?*

## How Chaos Testing Works

Chaos testing involves the simulation or injection of unusual events into the system. We should do this proactively—before these events have a chance to cause unscheduled downtimes or other impacts on the user experience.

Chaos testing works by hammering applications with unusual use cases, such as sending malformed inputs to a web app, overloading an app with traffic, or deliberately trying to trigger common vulnerabilities and exposures (CVEs) or well-known attacks like SQL injection.

Typically, we want to define key performance indicators (KPIs) to track the system's steady state in production. So, we define an acceptable blast radius before actively trying to break or disrupt the test target, so as not to cause a decline in user experience.

KPIs do differ, but typically the goals are to reduce the rate of failures caused by changes, reduce time spent putting out fires, and limit the duration of any downtime. As you might imagine, an effective monitoring system is important in these tests. For example, does the monitoring system alert key personnel before, during, and after threshold breaches? How about incident logs? Are they generated in real-time, are they tamper-proof, and do they catch all issues?

We might want to confirm that automated mitigation, such as horizontal and vertical scaling, works correctly in our CI/CD pipeline. Are more Virtual Machines (VMs) or containers spun up when there are increasing concurrent requests? Is more computing power applied to a VM in the event of a heightened and prolonged processing complexity? What happens when system clocks in financial workloads are



deliberately unsynchronized—does the system stop? Is the customer erroneously debited or credited? Are transaction receipts delivered late or not at all?

This sort of testing gives greater insight into the interventions or upgrades that could strengthen the system.

## Why Use Chaos Testing?

Try as we might, we cannot predict every production mishap. From an infrastructure misconfiguration, a single-line error from a developer, a slow microservice that impacts system-wide latency, or even simple human error—if something has the potential to go wrong, then it probably will. That is why we test, but why specifically use chaos testing?

## It Improves the Resilience of the System

One reason to use chaos testing is that it helps to determine resilience in production by deliberately experimenting with uncommon failures to see if the system's failback and failover mechanisms in place work. Typically, testing involves checking every issue your team usually encounters, excluding the unexpected. Chaos testing fills that hole and uses the information from your experiments to strengthen your system against such failures.

## It Reduces System Downtime

Chaos engineering helps you to understand system behavior during the event of a failure and helps to uncover the path to recovery of sub-systems. This means that you can swiftly figure out and possibly avoid or mitigate major

---

IT failures thereby reducing the loss of valuable production time, paying huge sums in damages, or impacting investor confidence.

### **You Get to Know the Weaknesses of the System**

Chaos testing is important because it generates new knowledge about the system's behaviors, properties, and performance. A distributed system usually tends to have more failure points due to its complexity and large-scale nature. Chaos testing tries to discover those failure points and identify what happens in the case of resource or object unavailability. In cases where you are hesitant to try new technologies because of reliability concerns, chaos testing is important because it identifies weak points and measures actual system behavior in real-time under the conditions in which you're interested.

### **Your Team Is Better Prepared**

For employers, an accidental benefit of chaos testing is that it reflects team incident response preparedness. The testing exercise is an opportunity to address process gaps and how emergency approvals work when needed, appraise technical knowledge and soft skills under pressure, and find out if you should retrain. This is especially important when your organization comes under statutory regulatory assessment for certification or endorsement.

### **It Improves Customer Satisfaction**

A final benefit of chaos testing is that it prevents service disruption by early identification of potential outages, which in turn improves the user experience.

### **How to Start Chaos Testing**

The first step for successful chaos testing is to acknowledge that you need it. Regardless

of the ability and foresight of your team, there are going to be unexpected issues arise with your system. Chaos testing is important for strengthening the resiliency of your system and giving you the confidence to know that whatever happens, your system responds well. Once your team understands the importance of Chaos testing, here's how you start.

You could start by using open source tools like [Chaos Monkey](#) or [ChaosBlade](#). Chaos Monkey only has the shutdown attack and a [spinnaker](#) and MySQL requirement. It works by sending a shutdown request to any random virtual machine (VM) in your architecture at any point within a set time. Before the attack launches, you might want to check whether there's an ongoing outage. To do this, you must write a custom Go script. This tool has severe limits for modern-day testing, which is why it is not popular.

In contrast, Chaos Blade provides multiple attack types—including resource consumption, packet loss, and more—for testing bare-metal, containers, and Kubernetes workloads. It also supports fault injection at the application level for C++, Java, and NodeJS applications. Examples of these types of faults are delayed code execution, arbitrary code insertion, and memory value modification. Sadly, it is not GUI-supported, the documentation is in Chinese, it requires coding knowledge, and the learning curve is steep.

The most prolific single chaos testing tool currently available is [Gremlin](#). It features a wide range of attack vectors that you can apply to VM, containers, and Kubernetes workloads at resource, state, and network strata over an intuitive GUI. For example, you can choose to simulate a state test for VM by selecting

preferred options on a web form, like killing a system process, changing system time, or doing an abrupt shutdown of the VM. Other tests for VM involve throttling resources like memory, CPU, and disk space, adding latency to matching traffic, or blocking access to DNS servers at the network layer.

The best way to properly test your system is to integrate chaos testing into your existing test suite. While chaos testing is great, it is only useful as a tool in your testing tool belt. Consider the [recent integration](#) of Gremlin to the [OpenText™ LoadRunner Professional](#) performance testing tool for example—it allows you to connect your Gremlin account via API keys to LoadRunner Professional and run Gremlin in-app. This allows you to add chaos testing to an already solid testing approach.

LoadRunner Professional is meant for use on-premises for local teams. It works by simulating Virtual Users (Vusers) which generate load by making application requests to your test target. The target must receive and acknowledge a response within a set timeframe to pass the performance test. If your team is globally distributed on-premises or is migrated to the cloud, the [OpenText™ LoadRunner Enterprise](#) and the [OpenText™ LoadRunner Cloud](#), respectively, are best suited for your needs.

Some examples of test targets on the LoadRunner family of performance testing tools are ERP apps like Oracle E-business or SAP, mobile, web, web 2.0, protocols like DNS, SMTP, FTP; Database (ODBC), and remote access (RDP, Citrix)—but there are many more.

### **Conclusion**

The essence of chaos testing is to strengthen system resilience. It's not meant to replace the testing you already do—instead, it

**Connect with Us**

[OpenText CEO Mark Barrenechea's blog](#)



complements your existing testing tools by finding bugs and vulnerabilities that companies usually miss.

- Increase service resiliency and ability to react to failures.
- Apply chaos principles continuously.
- Create and organize a central chaos engineering team.
- Follow best practices for chaos testing.

If you're ready to start chaos testing, consider [LoadRunner Professional](#), which integrates Gremlin for all your performance and chaos testing needs.

Learn more at

[www.microfocus.com/en-us/products/loadrunner-professional/overview](http://www.microfocus.com/en-us/products/loadrunner-professional/overview)

[www.microfocus.com/opentext](http://www.microfocus.com/opentext)