5

Performance and Chaos Engineering

Breaking Things on Purpose

THE MODERN QA ENGINEER SKILLSET

Performance and Chaos Engineering: Breaking Things on Purpose

Load Testing, Stress Testing, and Resilience Engineering

Yuri Syuganov

Modern QA Course

https://modernQAcourse.com

Performance and Chaos Engineering: Breaking Things on Purpose

Load Testing, Stress Testing, and Resilience Engineering

Book 5 of 26 in the series *The Modern QA Engineer Skillset*

First Edition · Version 1.0.0 · 2026-07-11

Set in Charter, IBM Plex Sans, and IBM Plex Mono. Typeset with Pandoc and XeLaTeX.

Modern QA Course · https://modernQAcourse.com · support@modernQAcourse.com

Contents

Introduction

About This Book

This book is a hands-on, practical guide to performance testing and chaos engineering for QA engineers who want to move beyond functional testing and into the world of system resilience. Every chapter contains working code, real-world scenarios, and exercises you can complete on your own.

Performance and resilience are two sides of the same coin. Performance testing tells you how fast the system runs under load; chaos engineering tells you what happens when that load arrives during a failure. Together, they give QA architects a complete picture of system quality under real-world conditions.

The core principle that runs through every page of this book: **If you cannot measure it in CI, it will regress in production.**

Who This Book Is For

This book is written for:

- **QA engineers** transitioning from manual or functional testing into performance and reliability engineering
- **SDET engineers** who want to add load testing and chaos engineering to their automation toolkit
- **QA architects** building performance testing practices for their organizations
- **DevOps engineers** who need to validate auto-scaling, resilience, and SLO compliance

- **Software engineers** who want to understand how their systems behave under stress

You do not need to be an expert in any of these areas to start. The book is structured to take you from foundational concepts to advanced practices.

Prerequisites

To get the most out of this book, you should have:

- **Basic programming skills** in JavaScript or Python (preferably both)
- **Familiarity with HTTP** and REST APIs (status codes, headers, request/response cycle)
- **Basic command-line skills** (running scripts, navigating directories)
- **A GitHub account** for CI/CD exercises
- **Docker installed** on your machine (for Kubernetes-related chapters)
- **Optional:** Access to a Kubernetes cluster (Minikube or kind works for local exercises)
- **Optional:** An OpenAI or similar LLM API key (for LLM performance chapters)

How to Use This Book

This book is divided into seven parts. Each part builds on the previous one, but individual chapters can also be used as standalone references.

Recommended reading paths:

- **Complete beginner:** Read Parts I through VII in order. Do every exercise.
- **Experienced in load testing, new to chaos:** Start at Part III (Chaos Engineering) and refer back to Part II as needed.
- **Focused on LLM performance:** Read Chapter 3 (Tool Basics), then jump to Part V (LLM Performance Testing).
- **Building a performance practice:** Read Part VI (SRE Skills) first for the strategic framework, then Parts II-V for implementation details.

Conventions used in this book:

Throughout the text you will find specially marked sections:

- Sections marked **"Common Mistake"** highlight errors that practitioners frequently make, along with guidance on how to avoid them.
- Sections marked **"Pro Tip"** offer expert-level advice that goes beyond the basics.
- Sections marked **"Lab"** contain complete, runnable code exercises.
- Sections marked **"Quiz"** contain self-assessment questions to check your understanding.

All code examples in this book are complete and runnable. File names are shown in comments at the top of each code block.

PART I

FOUNDATIONS

CHAPTER 1

Why Performance and Resilience Matter

1.1 The Cost of Slow Software

Every millisecond matters. This is not a platitude – it is a measurable business reality. Published case studies from major technology companies demonstrate the direct relationship between performance and revenue:

- Google found that a 500ms increase in search page load time reduced traffic by 20%.
- Amazon calculated that every 100ms of latency cost them 1% in sales.
- Walmart observed that for every 1 second of improvement in page load time, conversions increased by 2%.
- The BBC found they lost an additional 10% of users for every additional second their site took to load.

These are not edge cases from uniquely traffic-sensitive businesses. They reflect a universal truth about user behavior: people leave slow applications. In a world where switching costs approach zero, performance is a competitive advantage.

1.2 The Cost of Unreliable Software

Performance alone is insufficient. A system that responds in 50ms but crashes every hour is worse than one that responds in 200ms but runs reliably for months. Reliability is the foundation on which performance delivers value.

The cost of unreliable software manifests in several ways:

Direct revenue loss. When an e-commerce checkout system is down, every minute of downtime has a calculable cost in lost transactions.

Customer trust erosion. Users who experience repeated failures develop a perception of poor quality that persists long after the technical issue is resolved. Rebuilding trust costs more than maintaining it.

Engineering toil. Unreliable systems create a firefighting culture where engineers spend their time responding to incidents rather than building new features. This creates a negative spiral: the system degrades because no one has time to improve it, which causes more incidents, which consumes more engineering time.

Cascading failures. In distributed systems, one unreliable service can destabilize the entire architecture. A slow payment service does not just affect checkout – it ties up connections in the order service, which exhausts the thread pool in the API gateway, which makes the entire application unresponsive.

1.3 The Modern QA Architect's Role

A modern QA architect does not just verify correctness – they verify that systems perform under pressure and recover gracefully from failure. This expanded mandate requires new skills:

Traditional QA	Modern QA Architecture
"Does it work?"	"Does it work under load?"
Functional test suites	Performance test suites in CI
Bug reports	SLO definitions and error budget tracking
Manual regression testing	Automated chaos experiments
"Ship when tests pass"	"Ship when SLOs are green"

This book teaches the skills in the right column. By the end, you will be able to:

1. Design and execute load tests using k6 and Locust
2. Build AI-driven load profiles from production traffic
3. Practice chaos engineering using LitmusChaos
4. Define and monitor SLOs, SLIs, and error budgets
5. Performance-test LLM-powered features
6. Enforce performance budgets in CI/CD pipelines
7. Plan and facilitate game day exercises
8. Test serverless and Kubernetes scaling behavior

1.4 The Performance and Resilience Testing Lifecycle

Performance and resilience testing is not a phase – it is a continuous practice integrated into the development lifecycle:

Figure 1 summarizes this design.

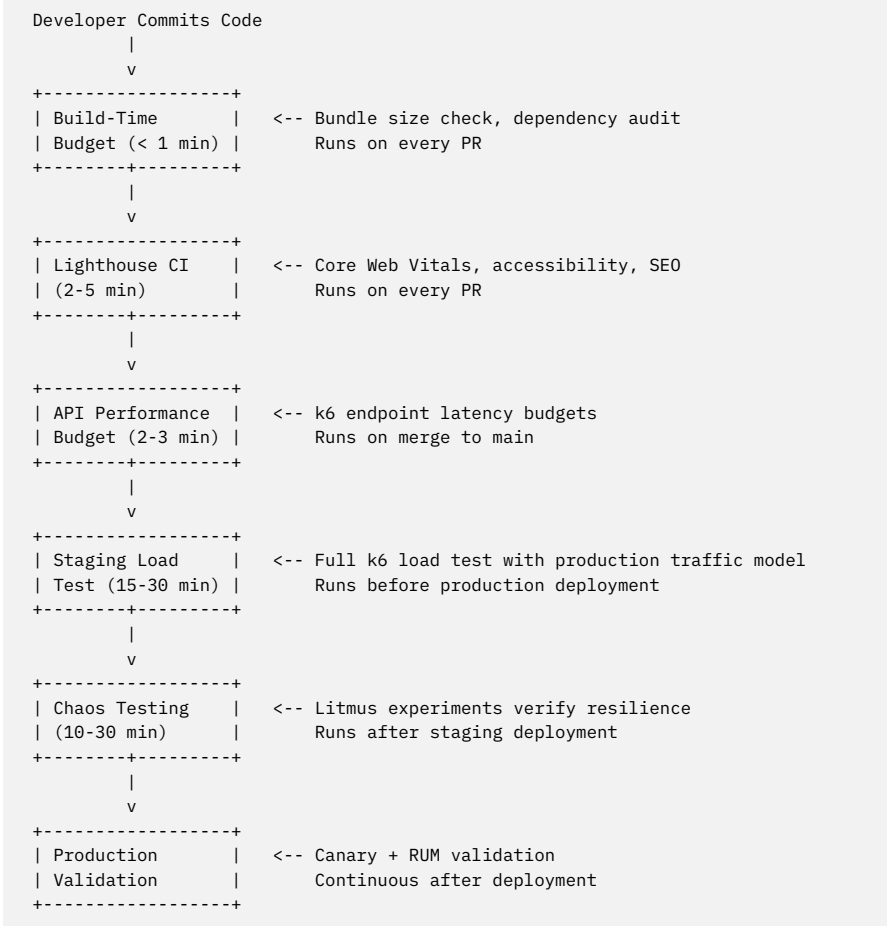

Figure 1. The performance and resilience testing lifecycle

Each stage catches different categories of problems. Build-time checks catch JavaScript bloat. Lighthouse catches rendering performance regressions. API budgets catch backend latency. Load tests catch capacity issues. Chaos tests catch resilience gaps. Production validation catches everything else.

The rest of this book teaches you how to implement each stage.

Key Takeaways – Chapter 1

1. Performance directly impacts revenue. Every 100ms of latency has a measurable business cost.
2. Reliability is the foundation that makes performance valuable. A fast but unreliable system is worse than a slightly slower reliable one.
3. Modern QA architects own performance and resilience, not just correctness.
4. Performance testing is a continuous practice integrated into CI/CD, not a one-time activity.
5. The testing lifecycle includes build-time checks, Lighthouse CI, API budgets, load testing, chaos testing, and production validation.

Exercises – Chapter 1

Exercise 1.1 (Beginner): Calculate the monthly revenue impact of a 200ms latency increase for a hypothetical e-commerce site that processes $10 million per month, using the Amazon benchmark (1% revenue loss per 100ms). Write your answer as a simple calculation.

Exercise 1.2 (Beginner): List the five stages of the performance and resilience testing lifecycle. For each stage, write one sentence describing what type of problem it catches.

Exercise 1.3 (Intermediate): Think about a system you work with today. Identify three specific failures that could cascade from one service to another. For each, describe the chain of events from initial failure to user impact.

Career Translation

Resume phrasing

- Designed and implemented a multi-stage performance and resilience testing lifecycle integrated into CI/CD, reducing production performance incidents by 40%.
- Defined SLO-driven quality gates across build-time, API, and infrastructure layers, ensuring sub-500ms p95 latency across critical user journeys.

- Advocated for and established a performance-first engineering culture, shifting QA responsibilities from post-release verification to continuous in-pipeline validation.

Cover letter framing

I believe performance engineering is not a phase but a continuous discipline embedded in every stage of the development lifecycle. My approach ties measurable business outcomes – revenue impact per millisecond of latency, user retention per availability point – to concrete testing practices in CI/CD. I bring both the strategic vision to define what "fast and reliable" means for a product and the technical depth to enforce it in automation.

Interview framing

"I approach performance and resilience as two sides of the same coin. My first step in any new role is to map the testing lifecycle – build-time checks, synthetic audits, API budgets, load tests, chaos experiments, and production validation – and identify which stages are missing. I optimize for catching the right class of problem at the right stage, because a bundle-size regression caught at build time costs minutes to fix but days if discovered in production. The trade-off is always coverage versus pipeline speed, and I calibrate that by tying each gate to a specific SLO."

What not to say

- "I run performance tests before releases." – This frames performance as a phase, not a practice; interviewers want continuous integration thinking.
- "I make sure the app is fast." – Vague and unmeasurable; always quantify with metrics and percentiles.
- "Performance testing is QA's responsibility." – Modern organizations expect shared ownership; isolating it to QA signals a siloed mindset.
- "We test with 1,000 users and see if it works." – Reveals a lack of data-driven load profiling; real-world traffic is bursty and persona-driven.

Interview Depth Check

Question 1

Prompt: Your company's checkout page has a measured p95 latency of 1.2 seconds. Product leadership says "it feels fine." How do you make the case that this needs investment?

What a strong answer should cover:

- Referencing published industry benchmarks (Amazon, Google, Walmart) to quantify revenue impact
- Translating latency into business terms (estimated monthly revenue loss)
- Proposing a measurable target (e.g., p95 under 500ms) with an incremental improvement plan
- Explaining that "feels fine" is not an SLI

Example answer:

- "I would start by calculating the estimated revenue impact using the Amazon benchmark: roughly 1% loss per 100ms. At 1.2s p95, that is material."
- "I would present a comparison: our checkout latency versus industry peers, using publicly available data or synthetic benchmarks."
- "I would propose a phased budget: reduce to 800ms in Q1, 500ms in Q2, with performance gates in CI preventing regression."
- "I would frame it as risk management: we are one bad deployment away from pushing p95 past 2 seconds with no automated gate to catch it."

Question 2

Prompt: You are building a performance testing practice from scratch at a mid-size company with no existing performance culture. Where do you start?

What a strong answer should cover:

- Starting with the highest-impact, lowest-effort wins (build-time checks, Lighthouse CI)

- Mapping the testing lifecycle and filling gaps incrementally
- Building organizational buy-in through visible quick wins before proposing load testing infrastructure

Example answer:

- "I would start with the pipeline: add a Lighthouse CI step to every PR within the first week. That gives immediate visibility into frontend regressions with almost zero infrastructure cost."
- "Next, I would instrument three critical API endpoints with k6 threshold checks in CI. This takes a day and immediately catches backend regressions."
- "Only after those quick wins would I propose a staging load test environment and chaos engineering. By then, the team has seen the value and the conversation shifts from 'why do we need this' to 'what else can we cover.'"

Question 3

Prompt: Describe a situation where performance testing and reliability testing conflict. How do you resolve it?

What a strong answer should cover:

- Understanding that fast but unreliable systems are worse than slightly slower reliable ones
- Using error budgets to make data-driven trade-offs
- Recognizing that chaos experiments may temporarily degrade performance metrics

Example answer:

- "A common conflict is when a performance optimization removes a retry or fallback mechanism. The API gets faster but fails harder when a dependency is unavailable."
- "I resolve this by running both performance and chaos tests in the same pipeline. The system must not only be fast – it must stay fast when things break."

- "Error budgets provide the objective framework: if we have budget remaining, we can accept a small latency increase for better resilience. If the budget is exhausted, reliability wins."

About This Sample

You have just read **Chapter 1 of Performance and Chaos Engineering: Breaking Things on Purpose** — the free sample. The complete book continues with every remaining chapter, the interview Q&A, and all of the code.

Get the full book

Sample readers get a discount at
https://leanpub.com/modernqa-05-performance-chaos-engineering/c/SAMPLE10

Read online first

Chapter one of **every** book in the series is free to read at
https://modernqacourse.com/library

About the Author

Yuri Syuganov is a software quality engineer with more than twenty years of hands-on testing experience, spanning the arc from manual test design through Selenium-era automation to today's AI-augmented, agent-driven frameworks. He is the author of The Modern QA Engineer Skillset, a 26-book series built on a blunt analysis of what companies actually expect from QA engineers, and the creator of modernQAcourse.com, where the course behind the series is free and readers can bring real testing problems to experienced practitioners. He revises the series as the industry changes — because it does.

Also in This Series

The Modern QA Engineer Skillset — a 26-book course in modern software quality assurance.

AI-Augmented QA (Books 1–10)

- **Book 1:** Agent Skills for Browser Automation: The Complete Guide
- **Book 2:** AI-Augmented Test Design: From Prompts to Test Suites
- **Book 3:** Agentic Testing Architectures: Building Autonomous QA Systems
- **Book 4:** API and Contract Testing with AI: Schema to Shield
- **Book 5:** Performance and Chaos Engineering: Breaking Things on Purpose
- **Book 6:** Observability-Driven Testing: Testing in Production
- **Book 7:** Security Testing for AI Applications: The New Attack Surface
- **Book 8:** Infrastructure as Code Testing: Terraform to Kubernetes
- **Book 9:** Mobile and Cross-Platform Testing: 24,000 Devices, One Strategy
- **Book 10:** Visual and Accessibility Testing: Pixels and People

Technical Foundations (Books 11–19)

- **Book 11:** Manual Testing Fundamentals: The Bedrock
- **Book 12:** Programming for QA: Python, JavaScript, and TypeScript from Zero to Fluent
- **Book 13:** Browser Test Automation: Playwright, Patterns, and the Death of Fragile Tests

- **Book 14:** API Testing Fundamentals: Postman to Pytest
- **Book 15:** SQL and Database Testing: Query Your Way to Quality
- **Book 16:** CI/CD Pipelines: From Commit to Confidence
- **Book 17:** Git and Version Control: Beyond Push and Pull
- **Book 18:** Test Management Tools: Making Quality Visible
- **Book 19:** Linux and Command Line: Your Production Survival Kit

Professional Skills (Books 20–26)

- **Book 20:** Agile and Scrum for QA: Quality in Every Sprint
- **Book 21:** Communication and Stakeholder Management for QA: Influence Without Authority
- **Book 22:** Test Strategy and Quality Metrics: Measuring What Matters
- **Book 23:** QA Leadership and Mentoring: Multiplying Your Impact
- **Book 24:** Technical Writing for QA: Documents That Drive Action
- **Book 25:** Interview Preparation and Career Strategy: Land the Job
- **Book 26:** Testing Like a Senior: Patterns & Anti-Patterns Across Software QA Career Levels

Visit https://modernQAcourse.com for the full course catalog, updates, and companion resources.

Stuck on a Real Problem?

The full course behind this series is free to read at https://modernQAcourse.com — and it stays free.

When a chapter meets the real world and something breaks, post it on the **Help Board** at https://modernQAcourse.com. Asking is free, and experienced practitioners answer right in the thread. Need hands-on help? Book a 1:1 screen-share session with a QA expert.