

# *contents*

---

<i>foreword</i>	xv
<i>foreword</i>	xvii
<i>preface</i>	xix
<i>acknowledgments</i>	xxi
<i>about this book</i>	xxiii
<i>about the author</i>	xxvi
<i>about the cover illustration</i>	xxvii

## **I** *Into the world of chaos engineering* 1

1.1	What is chaos engineering?	2
1.2	Motivations for chaos engineering	3
	<i>Estimating risk and cost, and setting SLIs, SLOs, and SLAs</i>	3
	<i>Testing a system as a whole</i>	5
	■ <i>Finding emergent properties</i>	5
1.3	Four steps to chaos engineering	6
	<i>Ensure observability</i>	9
	■ <i>Define a steady state</i>	10
	<i>Form a hypothesis</i>	10
	<i>Run the experiment and prove (or refute) your hypothesis</i>	11
1.4	What chaos engineering is not	11
1.5	A taste of chaos engineering	13
	<i>FizzBuzz as a service</i>	13
	■ <i>A long, dark night</i>	13
	<i>Postmortem</i>	14
	■ <i>Chaos engineering in a nutshell</i>	15

**PART 1 CHAOS ENGINEERING FUNDAMENTALS ..... 17****2 First cup of chaos and blast radius 19**

- 2.1 Setup: Working with the code in this book 20
- 2.2 Scenario 21
- 2.3 Linux forensics 101 22
  - Exit codes* 23 ▪ *Killing processes* 24 ▪ *Out-Of-Memory Killer* 26
- 2.4 The first chaos experiment 29
  - Ensure observability* 33 ▪ *Define a steady state* 34
  - Form a hypothesis* 34 ▪ *Run the experiment* 34
- 2.5 Blast radius 36
- 2.6 Digging deeper 38
  - Saving the world* 40

**3 Observability 43**

- 3.1 The app is slow 44
- 3.2 The USE method 45
- 3.3 Resources 47
  - System overview* 48 ▪ *Block I/O* 50 ▪ *Networking* 54
  - RAM* 59 ▪ *CPU* 66 ▪ *OS* 73
- 3.4 Application 75
  - cProfile* 76 ▪ *BCC and Python* 77
- 3.5 Automation: Using time series 79
  - Prometheus and Grafana* 80
- 3.6 Further reading 82

**4 Database trouble and testing in production 84**

- 4.1 We're doing WordPress 85
- 4.2 Weak links 86
  - Experiment 1: Slow disks* 87 ▪ *Experiment 2: Slow connection* 92
- 4.3 Testing in production 98

**PART 2 CHAOS ENGINEERING IN ACTION.....101****5****Poking Docker 103**

- 5.1 My (Dockerized) app is slow! 104
  - Architecture* 105
- 5.2 A brief history of Docker 106
  - Emulation, simulation, and virtualization* 106
  - Virtual machines and containers* 107
- 5.3 Linux containers and Docker 110
- 5.4 Peeking under Docker's hood 113
  - Uprooting processes with chroot* 114 □ *Implementing a simple container(-ish) part 1: Using chroot* 117 □ *Experiment 1: Can one container prevent another one from writing to disk?* 119
  - Isolating processes with Linux namespaces* 124 □ *Docker and namespaces* 127
- 5.5 Experiment 2: Killing processes in a different PID namespace 129
  - Implementing a simple container(-ish) part 2: Namespaces* 133
  - Limiting resource use of a process with cgroups* 135
- 5.6 Experiment 3: Using all the CPU you can find! 141
- 5.7 Experiment 4: Using too much RAM 143
  - Implementing a simple container(-ish) part 3: Cgroups* 146
- 5.8 Docker and networking 150
  - Capabilities and seccomp* 154
- 5.9 Docker demystified 157
- 5.10 Fixing my (Dockerized) app that's being slow 158
  - Booting up Meower* 158 □ *Why is the app slow?* 160
- 5.11 Experiment 5: Network slowness for containers with Pumba 161
  - Pumba: Docker chaos engineering tool* 161 □ *Chaos experiment implementation* 162
- 5.12 Other parts of the puzzle 166
  - Docker daemon restarts* 166 □ *Storage for image layers* 166
  - Advanced networking* 167 □ *Security* 167

## **6 Who you gonna call? Syscall-busters! 169**

- 6.1 Scenario: Congratulations on your promotion! 170  
*System X: If everyone is using it, but no one maintains it, is it abandonware?* 170
- 6.2 A brief refresher on syscalls 172  
*Finding out about syscalls* 174 ▪ *Using the standard C library and glibc* 176
- 6.3 How to observe a process's syscalls 178  
*strace and sleep* 178 ▪ *strace and System X* 182 ▪ *strace's problem: Overhead* 183 ▪ *BPF* 185 ▪ *Other options* 187
- 6.4 Blocking syscalls for fun and profit part 1: strace 188  
*Experiment 1: Breaking the close syscall* 189 ▪ *Experiment 2: Breaking the write syscall* 193
- 6.5 Blocking syscalls for fun and profit part 2: Seccomp 195  
*Seccomp the easy way with Docker* 196 ▪ *Seccomp the hard way with libseccomp* 198

## **7 Injecting failure into the JVM 201**

- 7.1 Scenario 202  
*Introducing FizzBuzzEnterpriseEdition* 202 ▪ *Looking around FizzBuzzEnterpriseEdition* 202
- 7.2 Chaos engineering and Java 204  
*Experiment idea* 204 ▪ *Experiment plan* 206 ▪ *Brief introduction to JVM bytecode* 207 ▪ *Experiment implementation* 215
- 7.3 Existing tools 222  
*Byteman* 223 ▪ *Byte-Monkey* 225 ▪ *Chaos Monkey for Spring Boot* 226
- 7.4 Further reading 227

## **8 Application-level fault injection 228**

- 8.1 Scenario 229  
*Implementation details: Before chaos* 230
- 8.2 Experiment 1: Redis latency 235  
*Experiment 1 plan* 235 ▪ *Experiment 1 steady state* 236  
*Experiment 1 implementation* 237 ▪ *Experiment 1 execution* 239 ▪ *Experiment 1 discussion* 240

8.3	Experiment 2: Failing requests	241	
	<i>Experiment 2 plan</i>	241 ▪ <i>Experiment 2 implementation</i>	242
	<i>Experiment 2 execution</i>	243	
8.4	Application vs. infrastructure	243	

## 9 *There's a monkey in my browser!* 246

9.1	Scenario	247		
	<i>Pgweb</i>	247 ▪ <i>Pgweb implementation details</i>	249	
9.2	Experiment 1: Adding latency	251		
	<i>Experiment 1 plan</i>	251 ▪ <i>Experiment 1 steady state</i>	252	
	<i>Experiment 1 implementation</i>	253 ▪ <i>Experiment 1 run</i>	255	
9.3	Experiment 2: Adding failure	256		
	<i>Experiment 2 implementation</i>	256 ▪ <i>Experiment 2 run</i>	258	
9.4	Other good-to-know topics	259		
	<i>Fetch API</i>	259 ▪ <i>Throttling</i>	260 ▪ <i>Tooling: Greasemonkey and Tampermonkey</i>	261

## PART 3 CHAOS ENGINEERING IN KUBERNETES.....263

### 10 *Chaos in Kubernetes* 265

10.1	Porting things onto Kubernetes	266			
	<i>High-Profile Project documentation</i>	267 ▪ <i>What's Goldpinger?</i>	268		
10.2	What's Kubernetes (in 7 minutes)?	268			
	<i>A very brief history of Kubernetes</i>	269 ▪ <i>What can Kubernetes do for you?</i>	270		
10.3	Setting up a Kubernetes cluster	272			
	<i>Using Minikube</i>	272 ▪ <i>Starting a cluster</i>	272		
10.4	Testing out software running on Kubernetes	274			
	<i>Running the ICANT Project</i>	274 ▪ <i>Experiment 1: Kill 50% of pods</i>	284 ▪ <i>Party trick: Kill pods in style</i>	289 ▪ <i>Experiment 2: Introduce network slowness</i>	290

### 11 *Automating Kubernetes experiments* 303

11.1	Automating chaos with PowerfulSeal	303	
	<i>What's PowerfulSeal?</i>	304 ▪ <i>PowerfulSeal installation</i>	306
	<i>Experiment 1b: Killing 50% of pods</i>	306 ▪ <i>Experiment 2b: Introducing network slowness</i>	308

11.2	Ongoing testing and service-level objectives	311
	<i>Experiment 3: Verifying pods are ready within (n) seconds of being created</i>	313
11.3	Cloud layer	318
	<i>Cloud provider APIs, availability zones</i>	319
	<i>Experiment 4: Taking VMs down</i>	321

## 12

### *Under the hood of Kubernetes* 324

12.1	Anatomy of a Kubernetes cluster and how to break it	324
	<i>Control plane</i>	325
	<i>Kubelet and pause container</i>	333
	<i>Kubernetes, Docker, and container runtimes</i>	335
	<i>Kubernetes networking</i>	338
12.2	Summary of key components	343

## 13

### *Chaos engineering (for) people* 345

13.1	Chaos engineering mindset	346
	<i>Failure is not a maybe: It will happen</i>	347
	<i>Failing early vs. failing late</i>	347
13.2	Getting buy-in	349
	<i>Management</i>	349
	<i>Team members</i>	350
	<i>Game days</i>	350
13.3	Teams as distributed systems	351
	<i>Finding knowledge single points of failure: Staycation</i>	353
	<i>Misinformation and trust within the team: Liar, Liar</i>	354
	<i>Bottlenecks in the team: Life in the Slow Lane</i>	355
	<i>Testing your processes: Inside Job</i>	356
13.4	Where to go from here?	357

appendix A	<i>Installing chaos engineering tools</i>	359
appendix B	<i>Answers to the pop quizzes</i>	367
appendix C	<i>Director's cut (aka the bloopers)</i>	375
appendix D	<i>Chaos-engineering recipes</i>	379
	<i>index</i>	385