index

Symbols

[interval] [count] 55
[tcpActiveOpens] 58
[tcpAttemptFails] 58
[tcpEstabResets] 58
[tcpInErrs] 58
[tcpInSegs] 58
[tcpOutRsts] 58
[tcpOutSegs] 58
[tcpPassiveOpens] 58
[tcpRetransSegs] 58
/api/does-not-exist endpoint 259
/api/v1/ endpoint 32–33
/healthz endpoint 315
./legacy_server command 172
./org/my subfolder structure 208
/search page 236
%CPU field 61
%ifutil field 56
~/src/examples/app command 234

A

ab command 91, 95, 97, 161–164, 189, 191, 194, 236–237, 239
accept syscall 182, 184
access syscall 179
active/s field 58
addTransformer method 213, 220
affinity 321
Agent class 212, 214
agentmain method 227
ALL keyword 156
alpine image 114
Alpine Linux 114
anti-affinity 321, 332
Apache Bench 31
Apache2 85
apache2 package 361
apache2-utils package 360
app=goldpinger label 278, 291, 296
AppKiller assault 226
application layer 75–79
  BCC suite and Python 77–79
  cProfile module 76–77
application-level fault injection 228–245
  application vs. infrastructure 243–244
  experiments
    failing requests 241–243
    Redis latency 235–240
  scenario 229–235
.apply(this, arguments) method 254
@app.route(231
assaults 226
atexit(3) function 177
attach_chaos_if_enabled function 238–239
automated monitoring systems 79–83
  Grafana 80–83
  Prometheus 80–83
autonomous mode 304
availability 326
available column 60
available field 61

B

bandwidth toxic 294
bash program 134
bc command 68–70, 74
BCC (BPF Compiler Collection)
  BPF and 185–187
  Python and 77–79
benchmarking 5
biotop tool 53–54
biotop-bpfcc command 54
black boxes 22
blast radius 19, 36, 349
block I/O (block input/output) devices 50–54
  biotop tool 53–54
  df tool 51–52
  iostat tool 52–53
BPF (Berkeley Packet Filter) 53, 185–187
-bpfcc 53, 185
bpfcc-tool package 54
br-b1ac9b3f5294 interface 152
bridge mode 150
bridge option 150
brk syscall 179
buffers 60
bytecode 207–215
  -javaagent 211–215
  reading 208–210
Byteman 223–225
  installing 223
  using 223–225

Byte-Monkey 225–226
  installing 225
  using 225–226

C

CACHE_CLIENT variable 237–239, 241
capabilities 155–157
--cap-add flag 156
--cap-drop ALL flag 156
--cap-drop flag 156
CAP_KILL 155
cap_sys_chroot 155
CAP_SYS_TIME 155
cat /proc/loadavg command 49
.catch handler 259
catch method 259
cereal_killer.sh script 35–36, 38
cgcreate 71, 136
cgexec 71, 136
cgroup.procs 139
cgroups tool
  implementing simple containers with 146–149
  killing processes in namespace 135–140
chaos engineering 1–15, 345, 350–358
  buy-in 349–351
    game days 350–351
    management 349–350
    team members 350
  defined
    what it is 2–3
    what it isn't 11–12
  FaaS example 13–15
    all-night investigation into problem 13–14
    four steps 15
    overview of 13
    postmortem 14–15
  four steps of 6–11, 377
    experiment 11
    hypothesis 10–11
    observability 9
    steady state 10
  mindset for 346–348
    failing early vs. failing late 347–348
    failure will happen 347
  motivations for 3–6
    estimating risk and cost 3–5
    finding emergent properties 5–6
    setting SLIs, SLOs, and SLAs 3–5
    testing system as a whole 5
  problems with term 378
  teams as distributed systems 351–357
    bottlenecks 355–356
    misinformation and trust 354–355
    single points of failure 353–354
    testing processes 356–357
  tool installation 359–366
    Linux tools 360–363
    Minikube 364–366
    prerequisites for 359
    source code 364
    WordPress 363–364
  tools comparison 376
CHAOS environment variable 238, 242
Chaos Monkey for Spring Boot 226–227
chaos network 152
chaos pizza 382–384
  ingredients 383
  preparation 383–384
chaos proxy 299–300
ChaosClient class 238
CHAOS_DELAY_SECONDS variable 238
chaos-engineering recipes 379–384
  chaos pizza 382–384
  SRE burger 379–382
ChaosMachine 227
chroot tool
  implementing simple container 117–119
  uprooting processes 114–117
classfileBuffer 212
ClassFileTransformer interface 212
ClassInjector class 218–220
className 212
ClassNode class 218
ClassPrinter class 212–213
ClassReader instance 217
CLIENT_PORT_OVERRIDE environment variable 295
clock_nanosleep syscall 180–181
close syscall 189–193
  analysis 192–193
  implementation 191–192
  steady state 191
CLOSED state 58
CLOSE-WAIT state 58
cloud 375
cloud layer 318–323
  availability zones 319–321
  cloud provider APIs 319–321
  taking VMs down 321–323
ClusterRole 276
clusters 324–342
  control plane 325–332
    etcd 326–329
    kube-apiserver 329–330
    kube-controller-manager 330–331
    kube-scheduler 332
  Docker, and container runtimes 335–338
  networking 338–342
    ingress networking 342
    pod-to-pod networking 339–340
    service networking 340–342
  pause container 333–335
  setting up using Minikube 272
  starting 272–274
Cmd 165
CNCF (Cloud Native Computing Foundation) 269
CNI (Container Networking Interface) 339
coll/s field 57
command column 125
consensus 326
container image format 112
Container Runtime Interface (CRI) 336
container runtimes 112, 335, 343
ContainerCreating 286
containerd 336
CONTAINER_PID 154
containers
  Docker and 110–112, 335–338
  implementing with cgroups 146–149
  implementing with chroot tool 117–119
  implementing with namespaces 133–135
  network slowness for 161–165
    experiment implementation 162–165
    Pumba 161–162

containers *(continued)*
one container preventing another from writing to disk 119–124
pause container 333–335
virtual machines and 107–110
containers, defined 104
Content-type header 236
control container 119, 123–124
Control group (cname) 125
control groups 71, 135
control plane 271
cProfile module 76–77
CPU (central processing unit) 66–72
mpstat P ALL 1 tool 69–70
stress command 141–143
top tool 67–69
cpu controller 136
- -cpu flag 143
cpu type 147
cpu,cpuacct 136, 149
cpu.cfs_period_us 137, 143
cpu.cfs_period_us, cpu.cfs_quota_us 137
cpu.cfs_quota_us 137, 143
- -cpus flag 141
cpu.shares 137, 143
CRI (Container Runtime Interface) 336
CRI-O 336
cumtime 76
curl command 33, 97
curl package 360

D

daemon restarts 166
DaemonSet 279
databases 84–100
testing in production 98–100
WordPress weak link example 86–98
overview of 85–86
slow connection experiment 92–98
slow disks experiment 87–92
deployment type 330
DEV keyword 55
df -h command 123
df tool 51–52, 60
dispatchEvent method 257
dmesg tool 27, 49–50
DNS (Domain Name System) server 5
Docker 103–168
advanced networking 167
blocking syscalls with seccomp 196–197
chroot tool
implementing simple container 117–119
uprooting processes 114–117
container runtimes 335–338
containers 110–112
daemon restarts 166
experiments
CPU usage 141–143
killing processes in different PID namespace 129–140
network slowness for containers with Pumba 161–165
one container preventing another from writing to disk 119–124
RAM overuse 143–149
history of 106–110
emulation, simulation, and virtualization 106–107
virtual machines and containers 107–110
namespaces 127–129
isolating processes with 124–127
networking 150–157
capabilities 155–157
seccomp 157
security 167–168
slow app problem 104–105
solving problem 158–161
storage for image layers 166–167
docker command-line client 112
Docker Hub 112
docker images command 141
docker inspect command 116
docker inspect firstcontainer command 116
docker network subcommand 150
docker ps command 128
Docker Registry 112
docker run 156
docker stack deploy 158
docker stack ls command 160
docker utility 112
docker0 bridge interface 150
dockerd 112
docker.io package 360
Domain Name System (DNS) server 5
down toxic 294
downtime 4
- -driver flag 273
dtrace tool 75

E

-e inject flag 190
-e inject option 190–191
e2e (end-to-end) tests 5, 241
EACCES error 190
eBPF (extended Berkeley Packet Filter) 53, 185
ECHO_REQUEST datagram 94
ECHO_RESPONSE 94
EDEV keyword 56
EINVAL (Invalid argument) 183
emergent properties 5–6
emulation 106–107
Endpoints field 281
end-to-end (e2e) tests 5, 241
Entrypoint 165
error event 257
ERRORS section 192
ESTABLISHED state 58
estres/s field 58
etcd 326–329
ETCP keyword 57
eth0 interface 55–56
events 256
example service 342
Example1 class 209
Example1.java program 208
exception assault 226
exec variants 74, 172
execsnoop tool 74
execve 179
exit codes 23
_exit(2) syscall 177
exit(3) glibc syscall 177
exit_group 181
export CONTAINER_ID 138, 145
extended Berkeley Packet Filter (eBPF) 53, 185

F

f flag 24
FaaS (FizzBuzz as a Service)
  example 13–15, 19–42
  all-night investigation into problem 13–14
  blast radiuses 36–37
  four steps 15, 29–36
    experiment 15, 34–36
    hypothesis 15, 34
    observability 15, 33–34
    steady state 15, 34
  Linux forensics 22–28
    exit codes 23
    killing processes 24–25
    Out-of-Memory Killer 26–28
  overview of 13
  postmortem 14–15
  scenario 21–22
  solution 38–41
  source code 20–21
faas001_a 32
faas001_b 32, 41
Failed requests 34
failed state 313–314
failing early 348–350
failing late 348
failing requests
  experiment 241–243
  execution 243
  implementation 242–243
  plan 241–242
fail_timeout parameter 33
failure container 121–124
fallocate script 120
fault injection
  application-level 228–245
    application vs. infrastructure 243–244
    failing requests experiment 241–243
    Redis latency experiment 235–240
    scenario 229–235
  JavaScript 256–259
  JVM 201–227
    existing tools 222–227
    experiment 204–222
    scenario 202–204
fault mode 225
fault option 189
fault tolerance 326
FBEE (FizzBuzzEnterprise-Edition) 202
Fetch API 259–260
fetch method 259
file utility 127
filesystem bundle 337
filesystems feature 111
Firecracker 109, 337
FizzBuzzEnterpriseEdition
  example 202–204
  experiment idea 204–206
  experiment implementation 215–222
  experiment plan 206–207
  JVM bytecode 207–215
FizzBuzzEnterpriseEdition/lib subfolder 203
FizzBuzzEnterpriseEdition.jar file 203
flanneld daemon 339
flask.request.cookies.get method 231
free tool 60
freegames package 363
fstat 179
fsync 190
Ftrace 188
full virtualization 107
full-stack Python development 351
fuzzing 6

G

game days 350
generate_legacy.py script 171
get command 274
get method 237–238
get_insterests function 243
get_interests function 237, 241–242
getpcaps command 155–156
getpid syscall 196–199
getstatic instruction 210
ghost container 160
git package 360, 364
glibc 176
GNU C Library 176
Goldpinger 268
  creating YAML files 278–280
  deploying 280–284
goldpinger-chaos 300
goldpinger-clusterrole ClusterRole 277
goldpinger-rbac.yml file 280
goldpinger-serviceaccount ServiceAccount 277
goldpinger.yml file 280
Grafana 80–83
GraphDriver section 116
.GraphDriver.Data.MergedDir path 116
Greasemonkey 261
grep 36
groups 136
gVisor 109, 337

H

-h argument 60
-h flag 189, 299
hardware interrupts (hi) 67
hardware virtualization 107
hardware-assisted virtualization 109
--hdd n option 89
--hdd option 89
head -n1 285
Hello chaos message 214, 260
--help command 274
hi (hardware interrupts) 67
host option 150
hostname -I command 153
--human option 50
Hyper-V Requirements 366
hypervisors 108
hypothesis
  FaaS example 15, 34
  forming 10–11

I

-i flag 116
ICANT project 267
ICMP (Internet Control Message Protocol) 94
id (idle time) 67
if statement 240
image layer storage 166–167
Image Specification 337
imagePullPolicy: Always 318
Inception-style reality 110, 156
index function 230–231
info endpoint 249
ingress networking 342
ingress type 342
input/output (block I/O) devices 50
instrumentation package 211, 213
integration tests 5, 241
internal.jdk package 220

Internet Control Message Protocol (ICMP) 94
invokestatic instruction 216, 218
invokestatic JVM instruction 216
invokevirtual instruction 210
IOException 205–206, 215
iostat tool 52–53, 89
ip addr command 151
ip command 152
ipc (Interprocess Communication) 125
iseg/s field 58
isegerr/s field 58

J

java command 209
Java Management Extensions (JMX) 226
-javaagent argument 214, 223, 225
javaagent argument 219–220, 223
-javaagent package 211–215
-javaagent parameter 224
javac command-line tool 209
javacalls 204
java.io.IOException 224
java.io.PrintStream 210
java.lang.instrument interface 201, 207
java.lang.instrument package 211–212, 222, 227
java.lang.System class 210
javap -c org.my.Example1 command 209
javap tool 209
JMX (Java Management Extensions) 226
jQuery 250
JS (JavaScript) 246–262
  experiments
    adding failure 256–259
    adding latency 251–256
  Fetch API 259–260
  Greasemonkey and Tampermonkey 261
  pgweb 247–248
    implementation details 249–250
  scenario 247–250
  throttling 260–261
--json flag 126
JVM (Java Virtual Machine) 201–227
  existing tools 222–227
    Byteman 223–225
    Byte-Monkey 225–226
    Chaos Monkey for Spring Boot 226–227
  experiment 204–222
    bytecode 207–215
    finding right exception to throw 205–206
    idea 204–206
    implementation 215–222
    injecting code 216–220
    plan 206–207
  scenario 202–204

K

Kata Containers 109, 337
kernel space 173
kill command 22, 24–25, 28, 36, 172, 284
KILL signal 25
killer_while.sh script 38, 41
killing processes 24–25
  in different PID namespace 129–140
    cgroups tool 135–140
    implementing simple containers with namespaces 133–135
  Out-of-Memory Killer 26–28
known unknowns 45
kube-apiserver 329–330, 333, 339
kube-apiserver component 325, 330, 343
kube-apiserver, kube-controller-manager 330
kube-cloud-manager component 325
kube-controller-manager 330–331
kube-controller-manager component 325, 330, 332, 343
kubectl 284–285, 314, 365
kubectl --help command 274
kubectl apply command 314, 325, 332
kubectl apply -f goldpinger-chaos.yml command 298
kubectl apply -f goldpinger-rbac.yml command 280, 285
kubectl command 271, 283, 306–308, 317
kubectl config file 305, 307
kubectl configuration file 289
kubectl cp 290
kubectl delet 314
kubectl exec 290
kubectl get --help 274
kubectl get command 285
kubectl get pods --watch command 285–286, 313
kubectl get pods -A command 273
kubectl get pods command 282, 285, 307
KubeInvaders 289
kube-proxy component 341, 343
Kubernetes 265–302, 324–344
  anatomy of cluster 324–342
    control plane 325–332
    Docker, and container runtimes 335–338
    networking 338–342
    pause container 333–335
  automating experiments 303–323
    cloud layer 318–323
    ongoing testing and service-level objectives 311–318
    PowerfulSeal 303–311
  history of 269–270
  key components 343
  overview of 268–272
  porting onto 266–268
    Goldpinger 268
    project documentation 267
  setting up cluster 272–274
    starting cluster 272–274
    using Minikube 272
  terminology 274–275
  testing software running on 274–302
    killing pods experiment 284–290
    network slowness experiment 290–302
  running project 274–284
Kubernetes cluster 270
kube-scheduler component 325, 331–332, 343
kube-thanos.sh script 285–286
kworker 54

L

labels 278
latency 93–95
  JavaScript 251–256
  Redis 235–240
latency assault 226
latency toxic 294
latency type 309
layers 115
ldc instruction 210
ldd command 118
legacy image 197
legacy_server binary 171, 191
libseccomp 198–199
libseccomp-dev package 198
libseccomp-devel package 198
lightweight isolation 107
Linux 22–28, 272
  exit codes 23
  killing processes 24–25
  Out-of-Memory Killer 26–28
Linux containers 110
LISTEN state 58
lo network interface 55
load averages 49
LowerDir 116
lsns command 125, 127, 129, 153
-lwxrq pod 286

M

main method 208–209, 213–216
man 2 read command 176
man 2 syscall-name 181
man 2 syscalls command 175
man 3 read 177
man 8 ld.so 179
man cgroups 140, 158
man command 175
man man command 174
man namespaces 158
man proc 49
man ps command 132
man sar 56
man strace(1) 183
man tc 93
man top 61
man unshare 133
manpages package 361
manpages-posix package 361
manpages-posix-dev package 361
max_fails 33
mean time between failure (MTBF) 347
mean time to failure (MTTF) 319
memory assault 226
memory cgroup 149
memory controller 136
- -memory flag 143
memory utilization 332
memory.limit_in_bytes value 139–140
memory.usage_in_bytes value 139–140
meower 160
MergedDir 116
Minikube
  installing 364–366
    Linux 364–365
    macOS 365–366
    Windows 366
  setting up clusters 272
minikube service command 282, 299
minikube service goldpinger command 282, 297, 310
minikube start - -driver 273
minikube start command 273
minikube stop 273
mmap syscall 179–180
Mounts (mnt) 125
mprotect syscall 180
mpstat 66, 70
mpstat P ALL 1 tool 69–70
mpstat -u -P ALL 2 command 142
MTBF (mean time between failure) 347
MTTF (mean time to failure) 319
munmap 180
musl libc 177
my_container container 162
mynamespace namespace 322, 342
MySQL 85
mystery001 program 26–27, 65
mystery002 command 68, 70, 74
mystery002 script 71

N

-n1 flag 26
Name member 116
namespaces
  Docker and 127–129
  isolating processes with 124–127
  killing processes in 129–140
    cgroups tool 135–140
    implementing simple containers with namespaces 133–135
nanosleep syscalls 179
ncalls 76
net (Network) 125
net namespaces 153–154
netem subcommand 162
- -network host container 150
- -network none 150
networking
  Docker 150–157
    advanced networking 167
    capabilities 155–157
    seccomp 157
  Kubernetes 338–342
    ingress networking 342
    pod-to-pod networking 339–340
    service networking 340–342
  network interfaces 54–59
    sar tool 55–58
    tcptop tool 58–59
  network slowness and PowerfulSeal 308–311
  network slowness for containers 161–165
    experiment implementation 162–165
    Pumba 161–162
  slowness 290–302
Networking feature 111
new Event('timeout') 257
new-filesystem.sh script 134
NEW_FS_FOLDER 156
NFS (Network File System) 193
nginx package 35, 360
ni (nice time) 67
niceness 27, 70
nodeAction 322
node_cpu_seconds_total CPU metric 81
Node.js 377
none option 150
NotReady 334
NPROCS 125
nr_throttled 143
NS column 126
nsenter command 131, 154

NULL argument 179
null driver 150
number of nines 4

O

-o flag 132
-o name flag 285
observability 9, 15
  ensuring 9
  FaaS example 15, 33–34
  slow app problem 43–83
    application layer 75–79
    automated monitoring systems 79–83
    overview of 44–45
    resources 47–75
    solving problem 70–72
    USE method 45–47
  syscalls 178–188
    BPF 185–187
    Ftrace 188
    strace tool 178–184
    SystemTap 188
OCI image 337
OOM (Out-of-Memory) Killer 25–28
oom_dump_tasks 29
oomkill tool 65–66
--oom-kill-disable flag 146
oom_reaper 27
oom_score_adj column 27
open syscall 172, 179
openat 179
openjdk-8-jdk package 361
opensnoop tool 73–74
operand stack 210
org/my/Example1 class 214
org.agent package 217
org.agent2 package 217
org.agent2.Agent 220
org.agent2.ClassInjector class 217
org.agent.Agent 214
org.my.Example1 class 209
orsts/s field 58
OS (operating system) 73–75
  execsnoop tool 74
  opensnoop tool 73–74
oseg/s field 58
OS-level virtualization 107
out static field 210
output method 206, 221, 224
overlay2 116
OXIPROXY_URL 299

P

-p flag 187
PACKAGE 360
passive/s field 58
PATH 294, 361, 366
pause container 333–335, 339
pending state 282, 286, 313–314
perf tool 75
pgweb 247–248, 363
  implementation details 249–250
  installing 362
pgweb --user 247
php package 361
PID (process ID) 36, 125
--pid flag 134
pid namespace 127, 129
ping command 94
Pip dependencies 363
platform virtualization 107
podAction 307, 322
pods 274
  killing
    killing half experiment 284–288, 306–308
    tools for 289–290
  pod-to-pod networking 339–340
  verifying pods are ready 313–318
pod-to-pod networking 339–340
policy file 304
PORT environment variable 279, 295
portability 173
postgresql package 361
PowerfulSeal 303–311
  defined 304–305
  installing 306
  killing pods experiment 306–308
  network slowness experiment 308–311
powerfulseal --help command 306
powerfulseal autonomous --policy-file experiment1b.yml command 307
powerfulseal autonomous --policy-file experiment2b.yml command 310
powerfulseal command 306
powerfulseal pip package 306
powerfulseal/powerfulseal image 306
pquota option 124
premain class 220
premain method 212–214, 219–220
Premain-Class attribute 212–214, 219
preparing dependencies 314
println call 208
.println method 210
prio qdisc 96
--privileged flag 167
probeHTTP 314, 316
process ID (PID) 36, 125
production, testing in 98–99
profile module 76
profile.json 196
Prometheus 80–83
Promise object 259
prom.yml configuration file 80
PROT_NONE flag 180
protocol 112
PROT_READ flag 180
ps command 34, 132, 148
ptrace default leverage 337
ptrace syscall 184
Pumba 161–162, 361
pumba help 161
pumba netem command 162
Python
  BCC suite and 77–79
  installing 362
python3-pip package 360
pythonflow 75, 77–78
pythonstat 75, 77–78

Q

QA (quality assurance) environment 37
qdisc 93
queueing discipline 93

R

raise_rediserror_every_other_time_if_enabled function 242
RAM (random access memory) 59–66
  free tool 60
  oomkill tool 65–66
  stress command 143–149

RAM (random access memory) *(continued)*
top tool 60–63
vmstat tool 63–65
read syscall 172, 175, 179, 183, 186
read-eval-print loop (REPL) 76
readlink command 127
recommend_other_products function 231
Redis latency experiment 235–240
discussion 240
execution 239–240
implementation 237–239
plan 235–236
steady state 236–237
RedisError 241
redis.exceptions.RedisError 242
redis-server command 234
REFRESH_PERIOD variable 284
REPL (read-eval-print loop) 76
ReplicaSet 330
reset function 231
resources 45, 47–75, 274
block I/O devices 50–54
biotop tool 53–54
df tool 51–52
iostat tool 52–53
CPU 66–72
mpstat P ALL 1 tool 69–70
top tool 67–69
network interfaces 54–59
sar tool 55–58
tcptop tool 58–59
OS 73–75
execsnoop tool 74
opensnoop tool 73–74
other tools 75
RAM 59–66
free tool 60
oomkill tool 65–66
top tool 60–63
vmstat tool 63–65
system overview 48–50
dmesg tool 49–50
uptime tool 48–49
response.set_cookie 231
retrans/s field 58
retval=<return code> argument 190
rkB/s field 52
rkt 336
--rm flag 114
--rm option 124
RST flag 58
runc 135
Running 281, 286
running state 313–314
Runtime Specification 337
runtimes 376
rxcmp/s field 56
rxdrop/s field 57
rxerr/s field 57
rxfifo/s field 57
rxfram/s field 57
rxkB/s field 56
rxmcst/s field 56
rxpck/s field 56

S

-S count flag 181
-s flag 64
sar tool 55–58
saturation 45
scenarios 304
SCMP_ACT_ERRNO default action 196
scraping metrics 80
search function 231
seccomp tool
blocking syscalls 195–199
with Docker 196–197
with libseccomp 198–199
Docker networking 157
seccomp_init function 198
seccomp_load function 198
seccomp_release function 198
seccomp_rule_add function 198
Secure Shell (SSH) 269
security feature 111, 173
--security-opt seccomp 157
selectors 278
send method 254–256
service networking 340–342
ServiceAccounts 276
service-level agreements (SLAs) 3–5
service-level indicators (SLIs) 3–5
service-level objectives (SLOs) 3–5
services 275
sessionID cookie 231
set method 237–238
set_session_id function 230
setTimeout function 254
si (software interrupts) 67
SIGFPE signal 23
signal= sig argument 190
SIGTERM 25
simulation 106–107
site reliability engineering (SRE) 3, 44
SLAs (service-level agreements) 3–5
sleep 1 command 178
sleep 3600 24
sleep command 24, 178
sleep process 138
slicer toxic 294
SLIs (service-level indicators) 3–5
SLOs (service-level objectives) 3–5
slow app problem 43–83
application layer 75–79
BCC suite and Python 77–79
cProfile module 76–77
automated monitoring systems 79–83
Grafana 80–83
Prometheus 80–83
Docker 104–105
architecture 105
CPU usage 141–143
killing processes in different PID namespace 129–140
network slowness for containers with Pumba 161–165
one container preventing another from writing to disk 119–124
RAM overuse 143–149
overview of 44–45
resources 47–75
block I/O devices 50–54
CPU 66–72
network interfaces 54–59
OS 73–75
RAM 59–66
system overview 48–50
solving problem 70–72, 158–161
USE method 45–47
slow close toxic 294
slow connection experiment 92–98
implementation 95–98
latency 93–95

slow disks experiment 87–92
  discussion 91–92
  implementation 88–91
software interrupts (si) 67
solid-state drives (SSDs) 90
sort - -random-sort 285
source code 20–21, 364
SPA (single-page application) 250
SRE ('ShRoomEee)
  burger 379–382
  assembling finished product 382
  hidden dependencies 381
  ingredients 380
  making patty 381–382
SRE (site reliability engineering) 3, 44
SSDs (solid-state drives) 90
SSH (Secure Shell) 269
st (steal time) 67
stage=dev label 278
stapbpf 188
StartLimitIntervalSec 40
Staycation 353
steady state 9, 15
  breaking close syscall 191
  breaking write syscall 194
  defining 10
  FaaS example 15, 34
  JavaScript latency 252
stopHost action 322
- -storage-opt size 124
store_interests function 231, 241
strace command 178, 181–182, 184–186, 190–191, 194–195
strace -h flag 190
strace output 178
strace tool 75, 182–183
  blocking syscalls 188–195
    breaking close syscall 189–193
    breaking write syscall 193–195
  overhead 183–184
  sleep command and 178–181
stress command 70, 74, 89–91, 141–144
stress package 360
string type 278
strong isolation 107
succeeded state 313
sudo apt-get install bpfcc-tools linux-headers-$(uname -r) command 185
sudo apt-get install git 360
sudo apt-get install PACKAGE 360
sudo command 36, 119, 128, 178
sudo lsns -t pid command 135
sudo password 316
sudo pip3 install redis 234
sudo service postgresql start command 247
sudo strace command 181
sudo strace -C command 182
sudo syscount-bpfcc command 185
swap 60–61
sy (system time) 67
SYN-RCVD state 58
SYN-SENT stat 58
SYN-SENT state 58
syscalls 169, 172–200
  blocking with seccomp 195–199
    with Docker 196–197
    with libseccomp 198–199
  blocking with strace 188–195
    breaking close syscall 189–193
    breaking write syscall 193–195
  finding out about 174–176
  observability 178–188
    BPF 185–187
    Ftrace 188
    strace tool 178–184
    SystemTap 188
  overview of 172–178
  scenario 170–172
  using standard C library and glibc 176–178
syscount command 186–187
syscount tool 185
syscount-bpfcc 187
sysstat package 55, 360
system calls 172
system calls. *See* syscalls
system overview 48–50
systemctl daemon 40
systemctl restart 36
systemd service 36, 38, 40, 269
systemd unit file 40
SystemOutFizzBuzzOutputStrategy class 206, 221, 224
SystemTap 188

T

Tampermonkey 261
- -target flag 154
- -task flag 126
tc (Traffic Control) 235
tc command 94, 98, 161–162, 165, 167, 290
tc tool 93
- -tc-image flag 162, 165
TCP keyword 57
tcptop tool 58–59
telnet 97
Terminating state 286
testing
  chaos engineering not replacement for other methods of 12
  in production 98–99
  of system as whole 5
.then handler 259
then method 259
throttled_time 143
throttling 260–261
throwIOException method 215, 217–218
throws keyword 205
Time (time) namespace 125
time command 97
time curl localhost/blog/ command 97
timeout toxic 294
top tool 60–64, 67–69
tottime 76
toxics 294
Toxiproxy 292–294
toxiproxy type 309
toxiproxy-cli command 295, 299
toxiproxy-cli list command 299
toxiproxy-cli toxic add command 300
transform method 211–212, 217
TripleAgent 227
tv_sec argument 180
txcarr/s field 57
txcmp/s field 56
txdrop/s field 57
txerr/s field 57
txfifo/s field 57
txkB/s field 56
txpck/s field 56

U

unified 136
UniK 109
union filesystem 115
union mount 115
unistd.h 176–177
unit tests 5
unknown state 313
unknown unknowns 45
unshare command 133–134, 147
UpperDir 116
uptime tool 48–49
--url flag 299
us (user time) 67
USDT (User Statically Defined Tracing) 77
USE (utilization, saturation, and errors) 45–47
User ID (user) 125
user space 173
utilization 45, 51
UTS (uts) 125

V

-verbose flag 209
VERSION command 360
vim package 360
virtual machines
  containers and 107–110
  taking down 321–323
virtualenv 306
virtualization 106–107
VM (virtual machine) image 20
vmstat command 63
vmstat -D 65
vmstat -d 65
vmstat -f 65
vmstat tool 63–65

W

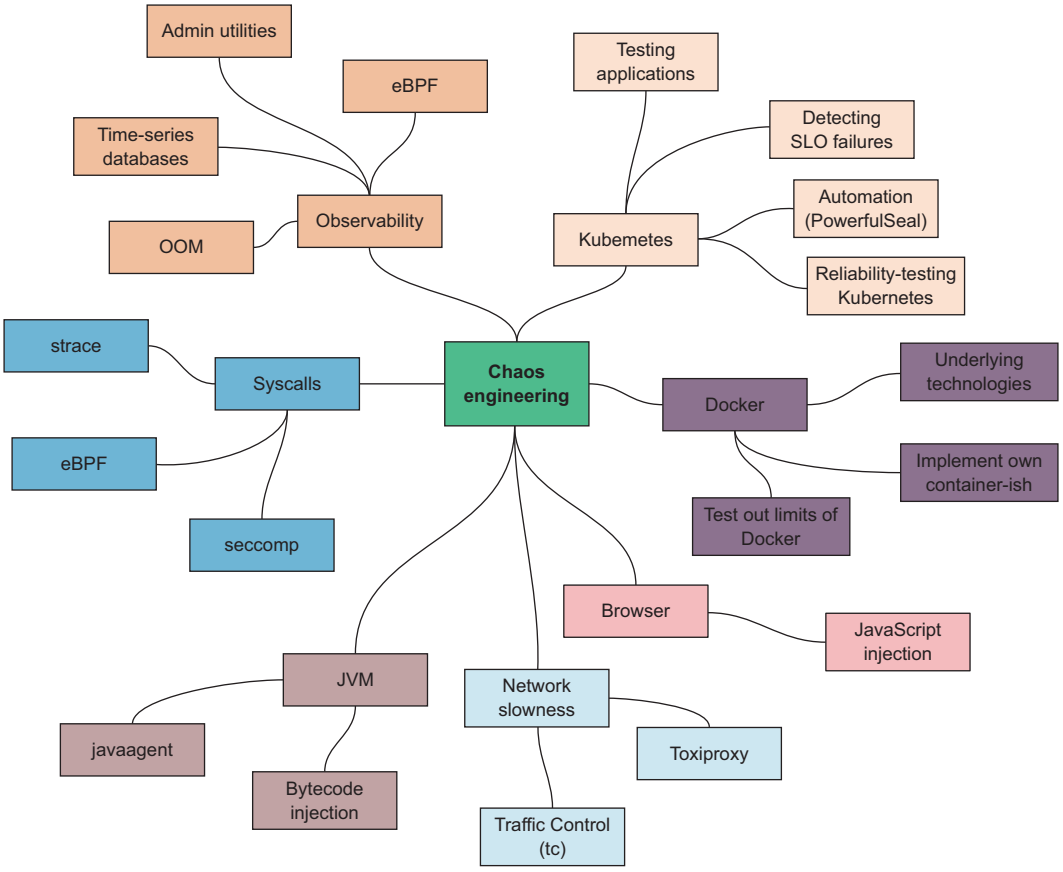
wa (I/O wait time) 67
--watch flag 285, 307
watch notification mechanism 330
wget command 59
window global scope 254
window variable 254
window.XMLHttpRequest.prototype.send 254, 257
--with-dtrace 78
wkB/s field 52
WORA (write once, run anywhere) principle 207
WordPress 85
wordpress package 361
WordPress weak link example 86–98
  configuring WordPress 363–364
  overview of 85–86
  slow connection experiment 92–98
    implementation 95–98
    latency 93–95
  slow disks experiment 87–92
    discussion 91–92
    implementation 88–91
write syscall 193–195
  implementation 194–195
  steady state 194

X

-x flag 52
-XDignore.symbol.file flag 220
xfs filesystem 124
XMLHttpRequest 253–254, 256–257, 259–260

Y

YAML files 278–280



TESTING/SOFTWARE ENGINEERING

Chaos Engineering

Mikolaj Pawlikowski

Can your network survive a devastating failure? Could an accident bring your day-to-day operations to a halt? Chaos engineering simulates infrastructure outages, component crashes, and other calamities to show how systems and staff respond. Testing systems in distress is the best way to ensure their future resilience, which is especially important for complex, large-scale applications with little room for downtime.

Chaos Engineering teaches you to design and execute controlled experiments that uncover hidden problems. Learn to inject system-shaking failures that disrupt system calls, networking, APIs, and Kubernetes-based microservices infrastructures. To help you practice, the book includes a downloadable Linux VM image with a suite of preconfigured tools so you can experiment quickly—without risk.

What's Inside

- Inject failure into processes, applications, and virtual machines
- Test software running on Kubernetes
- Work with both open source and legacy software
- Simulate database connection latency
- Test and improve your team's failure response

Assumes Linux servers. Basic scripting skills required.

Mikolaj Pawlikowski is a recognized authority on chaos engineering. He is the creator of the Kubernetes Chaos Engineering tool PowerfulSeal, and the networking visibility tool Goldpinger.

Register this print book to get free access to all ebook formats.
Visit https://www.manning.com/freebook

"The topics covered in this book are easy to follow and detailed. It provides a number of hands-on exercises to help the reader master chaos engineering."
—Kelum Prabath Senanayake
Echoworx

"The book we needed to improve our system's reliability and resilience."
—Hugo Cruz
People Driven Technology

"An important topic if you want to find hidden problems in your large system. This book gives a really good foundation."
—Yuri Kushch, Amazon

"One of the best books about in-depth infrastructure, troubleshooting complex systems, and chaos engineering that I've ever read."
—Lev Andelman
Terasky Cloud & Devops

ISBN: 978-1-61729-775-5

MANNING $49.99 / Can $65.99 [INCLUDING eBOOK]