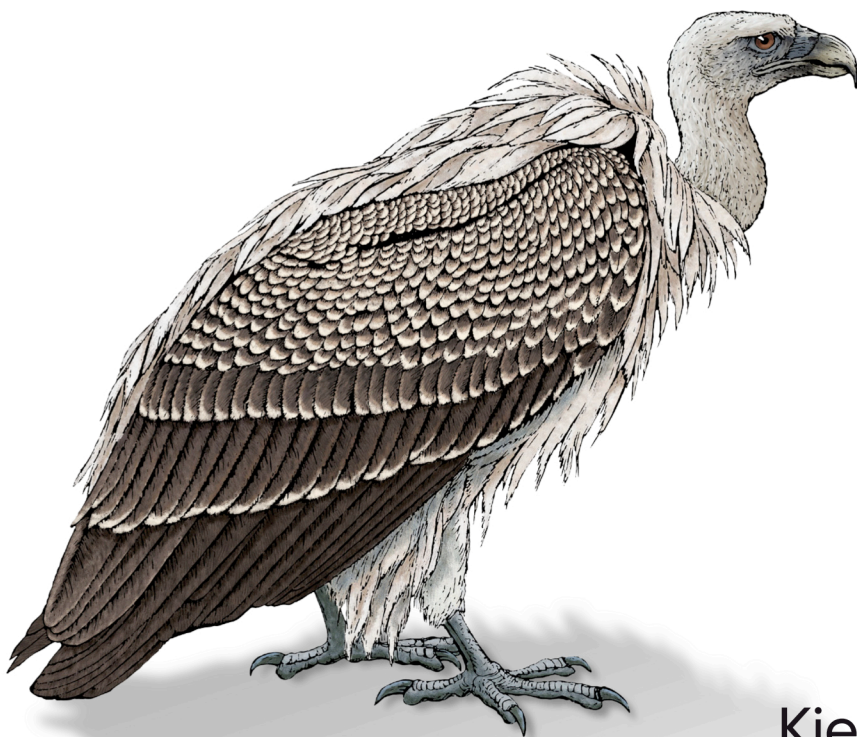


O'REILLY®

Übersetzung der
2. Auflage

Handbuch Infrastructure as Code

Prinzipien, Praktiken und Patterns für
eine cloudbasierte IT-Infrastruktur



Kief Morris

Übersetzung von Thomas Demmig

Papier
plus⁺
PDF.

Zu diesem Buch – sowie zu vielen weiteren O'Reilly-Büchern – können Sie auch das entsprechende E-Book im PDF-Format herunterladen. Werden Sie dazu einfach Mitglied bei oreilly.plus⁺:

www.oreilly.plus

Handbuch Infrastructure as Code

*Prinzipien, Praktiken und Patterns
für eine cloudbasierte IT-Infrastruktur*

Kief Morris

*Deutsche Übersetzung von
Thomas Demmig*

O'REILLY®

Kief Morris

Lektorat: Ariane Hesse

Übersetzung: Thomas Demmig

Fachliche Unterstützung: Pascal Euhus

Korrektorat: Claudia Lötschert, www.richtiger-text.de

Satz: mediaService, Gerhard Alfes, Siegen, www.mediaservice.tv

Herstellung: Stefanie Weidner

Umschlaggestaltung: Michael Oréal, www.oreal.de

Druck und Bindung: mediaprint solutions GmbH, 33100 Paderborn

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

ISBN:

Print 978-3-96009-170-7

PDF 978-3-96010-624-1

ePub 978-3-96010-625-8

mobi 978-3-96010-626-5

1. Auflage 2022

Translation Copyright für die deutschsprachige Ausgabe © 2022 dpunkt.verlag GmbH

Wieblinger Weg 17

69123 Heidelberg

Authorized German translation of the English edition *Infrastructure as Code: Dynamic Systems for the Cloud Age*, 2nd Edition, ISBN 9781098114671 © 2021 Kief Morris.

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

Dieses Buch erscheint in Kooperation mit O'Reilly Media, Inc. unter dem Imprint »O'REILLY«.

O'REILLY ist ein Markenzeichen und eine eingetragene Marke von O'Reilly Media, Inc. und wird mit Einwilligung des Eigentümers verwendet.

Hinweis:

Dieses Buch wurde auf PEFC-zertifiziertem Papier aus nachhaltiger Waldwirtschaft gedruckt. Der Umwelt zuliebe verzichten wir zusätzlich auf die Einschweißfolie.



Schreiben Sie uns:

Falls Sie Anregungen, Wünsche und Kommentare haben, lassen Sie es uns wissen: komentar@oreilly.de.

Die vorliegende Publikation ist urheberrechtlich geschützt. Alle Rechte vorbehalten. Die Verwendung der Texte und Abbildungen, auch auszugsweise, ist ohne die schriftliche Zustimmung des Verlags urheberrechtswidrig und daher strafbar. Dies gilt insbesondere für die Vervielfältigung, Übersetzung oder die Verwendung in elektronischen Systemen.

Es wird darauf hingewiesen, dass die im Buch verwendeten Soft- und Hardware-Bezeichnungen sowie Markennamen und Produktbezeichnungen der jeweiligen Firmen im Allgemeinen warenzeichen-, marken- oder patentrechtlichem Schutz unterliegen.

Alle Angaben und Programme in diesem Buch wurden mit größter Sorgfalt kontrolliert. Weder Autor noch Verlag noch Übersetzer können jedoch für Schäden haftbar gemacht werden, die in Zusammenhang mit der Verwendung dieses Buches stehen.

5 4 3 2 1 0

Vorwort	19
Warum ich dieses Buch geschrieben habe	20
Was in dieser Auflage neu und anders ist	20
Was kommt als Nächstes?	22
Was dieses Buch ist und was es nicht ist	22
Etwas Geschichte zu Infrastructure as Code	23
Für wen dieses Buch gedacht ist	24
Prinzipien, Praktiken und Patterns	24
Die ShopSpinner-Beispiele	25
In diesem Buch verwendete Konventionen	25
Danksagung	26

Teil I Grundlagen

1 Was ist Infrastructure as Code?	31
Aus der Eisenzeit in das Cloud-Zeitalter	33
Infrastructure as Code	34
Vorteile von Infrastructure as Code	34
Infrastructure as Code nutzen, um für Änderungen zu optimieren	35
Einwand »Wir haben gar nicht so häufig Änderungen, sodass sich Automation nicht lohnt«	35
Einwand »Wir sollten erst bauen und danach automatisieren«	36
Einwand »Wir müssen zwischen Geschwindigkeit und Qualität entscheiden«	37
Die Four Key Metrics	39
Drei zentrale Praktiken für Infrastructure as Code	40
Zentrale Praktik: Definieren Sie alles als Code	40
Zentrale Praktik: Kontinuierliches Testen und die gesamte aktuelle Arbeit ausliefern	41

Zentrale Praktik: Kleine einfache Elemente bauen, die Sie unabhängig voneinander ändern können	41
Zusammenfassung	42
2 Prinzipien der Infrastruktur im Cloud-Zeitalter	43
Prinzip: Gehen Sie davon aus, dass Systeme unzuverlässig sind.	44
Prinzip: Alles reproduzierbar machen.	44
Fallstrick: Snowflake-Systeme.	45
Prinzip: Erstellen Sie wegwerfbare Elemente	46
Prinzip: Variationen minimieren.	47
Konfigurationsdrift.	48
Prinzip: Stellen Sie sicher, dass Sie jeden Prozess wiederholen können .	50
Zusammenfassung	51
3 Infrastruktur-Plattformen	53
Die Elemente eines Infrastruktur-Systems	53
Dynamische Infrastruktur-Plattform	55
Infrastruktur-Ressourcen	57
Computing-Ressourcen	58
Storage-Ressourcen	59
Networking-Ressourcen.	61
Zusammenfassung	63
4 Zentrale Praktik: Definieren Sie alles als Code	65
Warum Sie Ihre Infrastruktur als Code definieren sollten	65
Was Sie als Code definieren können.	66
Wählen Sie Werkzeuge mit externalisierter Konfiguration aus	67
Managen Sie Ihren Code in einer Versionsverwaltung	67
Programmiersprachen für Infrastruktur	68
Infrastruktur-Scripting	70
Deklarative Infrastruktur-Sprachen	71
Programmierbare, imperative Infrastruktur-Sprachen.	73
Deklarative und imperative Sprachen für Infrastruktur.	74
Domänenspezifische Infrastruktur-Sprachen.	75
Allgemein nutzbare Sprachen und DSLs für die Infrastruktur.	76
Implementierungs-Prinzipien beim Definieren von	
Infrastructure as Code	77
Halten Sie deklarativen und imperativen Code voneinander getrennt	77
Behandeln Sie Infrastruktur-Code wie echten Code	77
Zusammenfassung	78

Teil II Arbeiten mit Infrastruktur-Stacks

5	Infrastruktur-Stacks als Code bauen	81
	Was ist ein Infrastruktur-Stack?	81
	Stack-Code	83
	Stack-Instanzen	83
	Server in einem Stack konfigurieren	83
	Low-Level-Infrastruktur-Sprachen	84
	High-Level-Infrastruktur-Sprachen	85
	Patterns und Antipatterns für das Strukturieren von Stacks	86
	Antipattern: Monolithic Stack	86
	Pattern: Application Group Stack	89
	Pattern: Service Stack	91
	Pattern: Micro Stack	92
	Zusammenfassung	93
6	Umgebungen mit Stacks bauen	95
	Worum es bei Umgebungen geht	95
	Auslieferungsumgebungen	96
	Mehrere Produktivumgebungen	96
	Umgebungen, Konsistenz und Konfiguration	97
	Patterns zum Bauen von Umgebungen	98
	Antipattern: Multiple-Environment Stack	99
	Antipattern: Copy-Paste Environments	100
	Pattern: Reusable Stack	102
	Umgebungen mit mehreren Stacks erstellen	104
	Zusammenfassung	106
7	Stack-Instanzen konfigurieren	107
	Eindeutige Kennungen durch Stack-Parameter erstellen	108
	Beispiel-Stack-Parameter	109
	Patterns zum Konfigurieren von Stacks	110
	Antipattern: Manual Stack Parameters	110
	Pattern: Stack Environment Variables	112
	Pattern: Scripted Parameters	114
	Pattern: Stack Configuration Files	117
	Pattern: Wrapper-Stack	120
	Pattern: Pipeline Stack Parameters	123
	Pattern: Stack Parameter Registry	126
	Konfigurations-Registry	129
	Eine Konfigurations-Registry implementieren	130
	Eine oder mehrere Konfigurations-Registries	132

Secrets als Parameter nutzen	133
Secrets verschlüsseln	133
Secretlose Autorisierung	133
Secrets zur Laufzeit injizieren	134
Wegwerf-Secrets	135
Zusammenfassung	135
8 Zentrale Praktik: Kontinuierlich testen und ausliefern	137
Warum Infrastruktur-Code kontinuierlich testen?	138
Was kontinuierliches Testen bedeutet	138
Was sollten wir bei der Infrastruktur testen?	140
Herausforderungen beim Testen von Infrastruktur-Code	143
Herausforderung: Tests für deklarativen Code haben häufig nur einen geringen Wert	143
Herausforderung: Das Testen von Infrastruktur-Code ist langsam .	146
Herausforderung: Abhängigkeiten verkomplizieren die Test-Infrastruktur	148
Progressives Testen	148
Testpyramide	149
Schweizer-Käse-Testmodell	151
Infrastruktur-Delivery-Pipelines	152
Pipeline-Stages	153
Scope von Komponenten, die in einer Stage getestet werden	154
Scope von Abhängigkeiten für eine Stage	155
Plattformelemente, die für eine Stage erforderlich sind	155
Software und Services für die Delivery-Pipeline	156
Testen in der Produktivumgebung	158
Was Sie außerhalb der Produktivumgebung nicht nachbauen können	159
Die Risiken beim Testen in der Produktivumgebung managen	160
Zusammenfassung	161
9 Infrastruktur-Stacks testen	163
Beispiel-Infrastruktur	163
Der Beispiel-Stack	164
Pipeline für den Beispiel-Stack	165
Offline-Test-Stages für Stacks	165
Syntax-Checks	166
Statische Offline-Code-Analyse	166
Statische Code-Analyse per API	167
Testen mit einer Mock-API	167

Online-Test-Stages für Stacks	168
Preview: Prüfen, welche Änderungen vorgenommen werden	169
Verifikation: Aussagen über Infrastruktur-Ressourcen treffen	169
Ergebnisse: Prüfen, dass die Infrastruktur korrekt arbeitet	171
Test-Fixtures für den Umgang mit Abhängigkeiten verwenden	172
Test-Doubles für Upstream-Abhängigkeiten	173
Test-Fixtures für Downstream-Abhängigkeiten	174
Komponenten refaktorisieren, um sie isolieren zu können	175
Lebenszyklus-Patterns für Testinstanzen von Stacks.	176
Pattern: Persistent Test Stack	176
Pattern: Ephemeral Test Stack	178
Antipattern: Dual Persistent and Ephemeral Stack Stages	179
Pattern: Periodic Stack Rebuild	180
Pattern: Continuous Stack Reset	182
Test-Orchestrierung.	183
Unterstützen Sie lokales Testen	184
Vermeiden Sie eine enge Kopplung mit Pipeline-Tools	185
Tools zur Test-Orchestrierung	185
Zusammenfassung.	185

Teil III Mit Servern und anderen Anwendungs-Laufzeitplattformen arbeiten

10 Anwendungs-Laufzeitumgebungen.	189
Cloud-native und anwendungsgesteuerte Infrastruktur	190
Ziele für eine Anwendungs-Laufzeitumgebung.	191
Deploybare Teile einer Anwendung.	191
Deployment-Pakete.	193
Anwendungen auf Server deployen	193
Anwendungen als Container verpacken	193
Anwendungen auf Server-Cluster deployen	195
Anwendungen auf Anwendungs-Cluster deployen.	195
Pakete zum Deployen von Anwendungen auf Cluster	197
FaaS-Serverless-Anwendungen deployen	198
Anwendungsdaten.	199
Datenschemata und -strukturen	199
Cloud-native Storage-Infrastruktur für Anwendungen.	200
Anwendungs-Connectivity	200
Service Discovery	201
Zusammenfassung.	203

11	Server als Code bauen	205
	Was gibt es auf einem Server	206
	Woher Dinge kommen	207
	Server-Konfigurationscode	209
	Code-Module für die Serverkonfiguration	210
	Code-Module für die Serverkonfiguration designen	210
	Server-Code versionieren und weitergeben	211
	Serverrollen	212
	Server-Code testen	213
	Server-Code progressiv testen	213
	Was Sie bei Server-Code testen	214
	Wie Sie Server-Code testen	215
	Eine neue Server-Instanz erstellen	216
	Eine neue Server-Instanz per Hand erstellen	217
	Einen Server mit einem Skript erstellen	218
	Einen Server mit einem Stack-Management-Tool erstellen	218
	Die Plattform für das automatische Erstellen von Servern konfigurieren	219
	Einen Server mit einem Network-Provisioning-Tool erstellen	220
	Server vorbereiten	221
	Hot-Cloning eines Servers	221
	Einen Server-Snapshot verwenden	222
	Ein sauberes Server-Image erstellen	222
	Eine neue Server-Instanz konfigurieren	223
	Eine Server-Instanz ausbacken	224
	Server-Images backen	225
	Backen und Ausbacken kombinieren	225
	Serverkonfiguration beim Erstellen eines Servers anwenden	226
	Zusammenfassung	227
12	Änderungen an Servern managen	229
	Patterns zum Changemanagement: Wann Änderungen angewendet werden	230
	Antipattern: Apply on Change	230
	Pattern: Continuous Configuration Synchronization	232
	Pattern: Immutable Server	234
	Wie Sie Serverkonfigurationscode anwenden	236
	Pattern: Push Server Configuration	236
	Pattern: Pull Server Configuration	238
	Andere Ereignisse im Lebenszyklus eines Servers	241
	Eine Server-Instanz stoppen und erneut starten	241
	Eine Server-Instanz ersetzen	242

Einen ausgefallenen Server wiederherstellen	243
Zusammenfassung	244
13 Server-Images als Code	245
Ein Server-Image bauen	246
Warum ein Server-Image bauen?	246
Wie Sie ein Server-Image bauen	247
Tools zum Bauen von Server-Images	247
Online Image Building	248
Offline Image Building	251
Ursprungsinhalte für ein Server-Image	252
Aus einem Stock-Server-Image bauen.	252
Ein Server-Image von Grund auf bauen	253
Herkunft eines Server-Image und seiner Inhalte.	253
Ein Server-Image ändern	254
Ein frisches Image aufwärmen oder backen	254
Ein Server-Image versionieren.	255
Server-Instanzen aktualisieren, wenn sich ein Image ändert	257
Ein Server-Image in mehreren Teams verwenden.	258
Umgang mit größeren Änderungen an einem Image	259
Eine Pipeline zum Testen und Ausliefern eines Server-Image verwenden.	259
Build-Stage für ein Server-Image.	260
Test-Stage für ein Server-Image	261
Delivery-Stages für ein Server-Image.	262
Mehrere Server-Images verwenden	263
Server-Images für unterschiedliche Infrastruktur-Plattformen	263
Server-Images für unterschiedliche Betriebssysteme.	264
Server-Images für unterschiedliche Hardware-Architekturen.	264
Server-Images für unterschiedliche Rollen	264
Server-Images in Schichten erstellen.	265
Code für mehrere Server-Images verwenden	266
Zusammenfassung	267
14 Cluster als Code bauen	269
Lösungen für Anwendungs-Cluster.	270
Cluster as a Service	270
Packaged Cluster Distribution	271
Stack-Topologien für Anwendungs-Cluster	272
Monolithischer Stack, der Cluster as a Service nutzt	273
Monolithischer Stack für eine Packaged-Cluster-Lösung.	274
Pipeline für einen monolithischen Anwendungs-Cluster-Stack	275
Beispiel für mehrere Stacks in einem Cluster	278

Strategien zur gemeinsamen Verwendung von Anwendungs-Clustern .	280
Ein großes Cluster für alles	281
Getrennte Cluster für Auslieferungs-Stages	282
Cluster für die Governance	283
Cluster für Teams.	284
Service Mesh	284
Infrastruktur für FaaS Serverless.	286
Zusammenfassung	288

Teil IV Infrastruktur designen

15 Zentrale Praktik: Kleine, einfache Elemente	291
Für Modularität designen	292
Eigenschaften gut designer Komponenten	292
Regeln für das Designen von Komponenten	293
Design-Entscheidungen durch Testen	296
Infrastruktur modularisieren	297
Stack-Komponenten versus Stacks als Komponenten	297
Einen Server in einem Stack verwenden.	299
Grenzen zwischen Komponenten ziehen	303
Grenzen mit natürlichen Änderungsmustern abstimmen	303
Grenzen mit Komponenten-Lebenszyklen abstimmen	304
Grenzen mit Organisationsstrukturen abstimmen	306
Grenzen schaffen, die Resilienz fördern.	307
Grenzen schaffen, die Skalierbarkeit ermöglichen	307
Grenzen auf Sicherheits- und Governance-Aspekte abstimmen	311
Zusammenfassung	312
16 Stacks aus Komponenten bauen	313
Infrastruktur-Sprachen für Stack-Komponenten	314
Deklarativen Code mit Modulen wiederverwenden	314
Stack-Elemente dynamisch mit Bibliotheken erstellen	315
Patterns für Stack-Komponenten	316
Pattern: Facade Module	317
Antipattern: Obfuscation Module	318
Antipattern: Unshared Module	320
Pattern: Bundle Module	321
Antipattern: Spaghetti Module	322
Pattern: Infrastructure Domain Entity	325
Eine Abstraktionsschicht bauen	327
Zusammenfassung	328

17 Stacks als Komponenten einsetzen	329
Abhängigkeiten zwischen Stacks erkennen	329
Pattern: Resource Matching	330
Pattern: Stack Data Lookup	333
Pattern: Integration Registry Lookup	336
Dependency Injection	339
Zusammenfassung	342

Teil V Infrastruktur bereitstellen

18 Infrastruktur-Code organisieren	345
Projekte und Repositories organisieren	345
Ein Repository oder viele?	346
Ein Repository für alles	346
Ein eigenes Repository für jedes Projekt (Microrepo)	349
Mehrere Repositories mit mehreren Projekten	350
Unterschiedliche Arten von Code organisieren	351
Projektsupport-Dateien	351
Projektübergreifende Tests	352
Dedizierte Projekte für Integrationstests	353
Code anhand des Domänenkonzepts organisieren	354
Dateien mit Konfigurationswerten organisieren	354
Infrastruktur- und Anwendungscode managen	356
Infrastruktur und Anwendungen ausliefern	356
Anwendungen mit Infrastruktur testen	358
Infrastruktur vor der Integration testen	359
Infrastruktur-Code zum Deployen von Anwendungen nutzen	359
Zusammenfassung	361
19 Infrastruktur-Code ausliefern	363
Auslieferungsprozess von Infrastruktur-Code	363
Ein Infrastruktur-Projekt bauen	364
Infrastruktur-Code als Artefakt verpacken	365
Infrastruktur-Code mit einem Repository ausliefern	365
Projekte integrieren	368
Pattern: Build-Time Project Integration	369
Pattern: Delivery-Time Project Integration	373
Pattern: Apply-Time Project Integration	375
Infrastruktur-Tools durch Skripte verpacken	378
Konfigurationswerte zusammenführen	379
Wrapper-Skripte vereinfachen	380
Zusammenfassung	381

20 Team-Workflows	383
Die Menschen	384
Wer schreibt Infrastruktur-Code?	386
Code auf Infrastruktur anwenden.	388
Code von Ihrem lokalen Rechner aus anwenden.	388
Code von einem zentralisierten Service anwenden lassen	389
Private Infrastruktur-Instanzen	391
Quellcode-Banches in Workflows	392
Konfigurationsdrift verhindern.	394
Automatisierungs-Verzögerung minimieren	394
Ad-hoc-Anwendung vermeiden	395
Code kontinuierlich anwenden	395
Immutable Infrastruktur	395
Governance in einem Pipeline-basierten Workflow	396
Zuständigkeiten neu ordnen	397
Shift Left.	398
Ein Beispielprozess für Infrastructure as Code mit Governance ...	398
Zusammenfassung	399
21 Infrastruktur sicher ändern	401
Reduzieren Sie den Umfang von Änderungen	402
Kleine Änderungen.	404
Refaktorisieren – ein Beispiel	406
Unvollständige Änderungen in die Produktivumgebung übernehmen .	407
Parallele Instanzen	408
Abwärtskompatible Transformationen	411
Feature Toggles	413
Live-Infrastruktur ändern	415
Infrastruktur-Chirurgie	417
Expand and Contract.	419
Zero-Downtime-Änderungen.	422
Kontinuität	423
Kontinuität durch das Verhindern von Fehlern.	424
Kontinuität durch schnelles Wiederherstellen.	425
Kontinuierliches Disaster Recovery	426
Chaos Engineering	427
Für Ausfälle planen	428
Datenkontinuität in einem sich ändernden System	430
Sperren	430
Aufteilen.	430

Replizieren	431
Neu laden	431
Ansätze zur Datenkontinuität mischen.	432
Zusammenfassung.	432
Index	433

Über dieses Buch

Die Praktiken im Bereich von Infrastructure as Code entwickelten sich vom Managen von Servern weiter zum Managen ganzer Stacks, aber diese neuen Möglichkeiten bringen auch zusätzliche Komplexität mit sich. Dieses Buch hilft Ihnen dabei, über die reinen Befehle hinaus ein Verständnis zu entwickeln und die Design-Patterns kennenzulernen, die hinter guter Praxis stehen. Zudem lernen Sie, wie Sie die nächste Stufe der Automatisierung meistern können.

– Patrick Debois, Begründer der DevOpsDays

Infrastructure as Code befindet sich an der Schnittstelle mehrerer Domänen des Software-Engineering. Dieses Buch ordnet die wichtigsten Best Practices aus jeder Domäne neu und führt sie auf ihre zentralen Aspekte zurück, sodass man beim Lesen einen Überblick über die Infrastruktur-Automatisierung erhält.

– Carlos Condé, VP of Engineering bei Sweetgreen

Ein unkomplizierter Ratgeber für die Orientierung in der Landschaft der Cloud-Infrastruktur mit Prinzipien, Praktiken und Patterns.

– Effy Elden, Technologist bei ThoughtWorks

Vor zehn Jahren schlug ich einem CIO bei einer weltweit agierenden Bank vor, sich mit Private-Cloud-Technologien und Werkzeugen zur Infrastruktur-Automation zu befassen, und er antwortete nur höhnisch: »Das mag ja für Start-ups ganz nett sein, aber wir sind zu groß und unsere Anforderungen sind zu komplex.« Und auch ein paar Jahre später wollten viele Unternehmen den Einsatz von Public Clouds noch nicht in Betracht ziehen.

Heutzutage ist Cloud-Technologie allgegenwärtig. Selbst die größten und unbeweglichsten Organisationen wechseln sehr zügig zu einer »Cloud First«-Technologie. Organisationen, die Public Clouds nicht einsetzen können, greifen auf dynamisch provisionierte Infrastruktur-Plattformen in ihren Data Centern zurück.¹ Die Möglichkeiten, die diese Plattformen bieten, werden so schnell so viel umfassender und besser, dass es schwer ist, sie zu ignorieren, ohne Gefahr zu laufen, stark ins Hintertreffen zu geraten.

Cloud- und Automatisierungs-Technologien reißen Barrieren nieder, die Änderungen an Produktivsystemen im Wege stehen, was allerdings wiederum zu neuen Herausforderungen führt. Während die meisten Organisationen ihre Änderungsgeschwindigkeit erhöhen wollen, können sie es sich nicht leisten, Risiken zu ignorieren oder sich der Gefahr auszusetzen, die Kontrolle über die Prozesse zu verlieren. Klassische Prozesse und Techniken für ein sicheres Ändern der Infrastruktur sind nicht auf häufige Veränderungen ausgelegt. Deren Arbeitsweise tendiert dazu, die Vorteile moderner Technologien des Cloud-Zeitalters auszubremsen und damit der Erledigung von Arbeit im Weg zu stehen und die Stabilität zu gefährden.²

1 So ist es beispielsweise vielen Regierungsorganisationen und Finanzunternehmen in Ländern ohne vorhandene Cloud gesetzlich untersagt, Daten oder Transaktionen im Ausland zu hosten.

2 Die von DORA im *State of DevOps Report* (<https://oreil.ly/ysk9n>) veröffentlichten Forschungsergebnisse zeigen, dass schwergewichtige Änderungsmanagement-Prozesse mit einer schlechten Performance bei Änderungsfehlerraten und anderen Messwerten mit Bezug zur Effektivität von Software Delivery korrelieren.

In Kapitel 1 nutze ich die Begriffe »Eisenzeit« und »Cloud-Zeitalter« (siehe »Aus der Eisenzeit in das Cloud-Zeitalter« auf Seite 33), um die unterschiedlichen Philosophien rund um das Managen »realer« Infrastruktur, bei der sich Fehler nur langwierig und teuer korrigieren lassen, und virtueller Infrastruktur, bei der Fehler schnell erkannt und behoben werden können, zu beschreiben.

Werkzeuge für Infrastructure as Code schaffen die Möglichkeit, so zu arbeiten, dass Änderungen häufiger, schneller und zuverlässiger ausgeliefert werden können, wodurch die Gesamtqualität Ihres Systems zunimmt. Aber die Vorteile entstehen nicht aus den Werkzeugen selbst, sondern daraus, wie Sie sie einsetzen. Der Trick ist, die Technologie als Hebel zu verwenden, um damit Qualität, Zuverlässigkeit und Compliance in den Änderungsprozess einzubringen.

Warum ich dieses Buch geschrieben habe

Die erste (englischsprachige) Auflage dieses Buchs schrieb ich, weil ich keine bündige Zusammenfassung für das Managen von Infrastructure as Code gefunden habe. Es gab viele Tipps, verteilt über Blog-Posts, Vorträge auf Konferenzen und Dokumentation von Produkten und Projekten. Aber für den Einsatz musste man alles durchforsten und sich selbst eine Strategie zusammenbasteln, wofür die meisten Leute einfach keine Zeit hatten.

Die Erfahrungen beim Schreiben der ersten Auflage waren überwältigend. Ich hatte die Gelegenheit, herumzureisen und mit Menschen überall auf der Welt über deren Erfahrungen zu sprechen. Diese Unterhaltungen verschafften mir neue Einblicke und stellten mich vor neue Herausforderungen. Ich lernte, dass der Wert des Schreibens eines Buchs, des Haltens von Vorträgen auf Konferenzen und des Beratens mit Kunden darin besteht, den fachlichen Austausch zu fördern. In der Branche sind wir immer noch dabei, unsere Ideen zu Infrastructure as Code zu sammeln, miteinander zu teilen und weiterzuentwickeln.

Was in dieser Auflage neu und anders ist

Seit die erste (englischsprachige) Auflage im Juni 2016 erschienen ist, hat sich ziemlich viel getan. Jene Auflage hatte den Untertitel »Managing Servers in the Cloud«, was die Tatsache widerspiegelte, dass sich damals ein Großteil der Infrastruktur-Automation um das Konfigurieren von Servern drehte. Seitdem sind Container und Cluster viel wichtiger geworden, und die Aktivitäten rund um die Infrastruktur haben sich hin zum Managen von Gruppen von Infrastruktur-Ressourcen bewegt, die auf Cloud-Plattformen provisioniert werden – was ich in diesem Buch als *Stacks* bezeichne.

Somit kümmert sich diese Auflage mehr um den Aufbau von Stacks, was das Verdienst von Tools wie CloudFormation oder Terraform ist. Ich gehe dabei von der Annahme aus, dass wir Stack-Management-Tools nutzen, um Infrastruktur-Ob-

jekte zusammenzufügen, die dann die Anwendungs-Laufzeitumgebungen bereitstellen. Zu diesen Laufzeitumgebungen können Server, Cluster und Serverless-Ausführungsumgebungen gehören.

Ich habe eine Menge geändert, weil ich seit der ersten Auflage viel gelernt habe über neue Herausforderungen und die Anforderungen von Teams beim Aufbau von Infrastruktur. Wie schon erwähnt sehe ich den Hauptvorteil von Infrastructure as Code darin, die Infrastruktur sicher und einfach anpassen zu können. Ich glaube, dass die Menschen deren Wichtigkeit unterschätzen, weil sie davon ausgehen, dass es sich bei Infrastruktur um etwas handelt, das sie einmal aufsetzen und dann vergessen.

Aber zu viele Teams, mit denen ich zu tun hatte, kämpfen damit, die Anforderungen ihrer Organisationen zu erfüllen – sie sind nicht dazu in der Lage, schnell genug zu wachsen und zu skalieren, die Geschwindigkeit der Softwareauslieferungen zu unterstützen oder die erwartete Zuverlässigkeit und Sicherheit zu bieten. Und wenn wir uns die Details ihrer Herausforderungen genauer anschauen, stellen wir fest, dass sie von dem Bedarf an Aktualisierungen, Korrekturen und Verbesserungen ihrer Systeme überrollt werden. Daher habe ich speziell diesen Bereich zum zentralen Thema dieses Buchs gemacht.

Diese Auflage stellt drei zentrale Praktiken für den Einsatz von Infrastructure as Code vor, die Änderungen sicher und einfach machen:

Definieren Sie alles als Code.

Das geht schon aus dem Namen hervor und sorgt für Reproduzierbarkeit und Konsistenz.

Testen Sie und liefern Sie kontinuierlich die aktuelle Arbeit aus.

Jede Änderung verbessert die Sicherheit. Sie ermöglicht es zudem, schneller und mit mehr Vertrauen voranzukommen.

Bauen Sie kleine, einfache Elemente, die Sie unabhängig voneinander ändern können.

Diese lassen sich einfacher und sicherer ändern als große Elemente.

Diese drei Praktiken unterstützen sich gegenseitig. Code lässt sich über die verschiedenen Stadien eines Änderungsmanagement-Prozesses hinweg einfach verfolgen, versionieren und ausliefern. Es ist einfacher, kontinuierlich kleinere Elemente zu testen. Kontinuierliches, eigenständiges Testen eines jeden Elements zwingt Sie dazu, ein lose gekoppeltes Design beizubehalten.

Solche Praktiken und die Details ihrer Anwendung sind uns aus der Welt der Softwareentwicklung vertraut. In der ersten Auflage dieses Buchs bezog ich mich auf Praktiken der agilen Softwareentwicklung und der Auslieferung. In dieser Auflage habe ich zudem auf Regeln und Praktiken effektiven Designs zurückgegriffen.

In den letzten Jahren habe ich gesehen, wie Teams bei größeren und komplizierteren Infrastruktur-Systemen ins Straucheln gerieten, und festgestellt, wie das Anwenden der Erfahrungen mit Software-Design-Patterns und -Prinzipien helfen konnte, daher habe ich eine Reihe von Kapiteln dazu aufgenommen.

Ich habe auch gesehen, dass das Organisieren von Infrastruktur-Code und die Arbeit damit für viele Teams schwierig ist, daher habe ich eine Reihe von kritischen Punkten angesprochen. Ich beschreibe, wie man eine Codebasis gut organisiert hält, wie man Entwicklungs- und Testinstanzen für die Infrastruktur bereitstellt und wie man die Zusammenarbeit mehrerer Personen managt – einschließlich derer, die für Governance verantwortlich sind.

Was kommt als Nächstes?

Ich glaube nicht, dass wir als Branche beim Umgang mit Infrastruktur schon ausgelernt haben. Ich hoffe, dieses Buch gibt einen guten Überblick über all das, was Teams heutzutage als effektiv ansehen. Und motiviert ein bisschen, es noch besser zu machen.

Ich erwarte, dass sich Toolchains und Vorgehensweise in fünf Jahren wieder weiterentwickelt haben. Vielleicht gibt es mehr universell einsetzbare Sprachen zum Bauen von Bibliotheken, und eventuell generieren wir Infrastruktur dynamisch, statt die statischen Umgebungsdetails auf niedriger Ebene zu definieren. Wir müssen mit ziemlicher Sicherheit beim Verwalten von Änderungen an Live-Infrastruktur besser werden. Die meisten mir bekannten Teams haben immer Angst, wenn sie Code auf Live-Infrastruktur anwenden. (Ein Team bezeichnet Terraform als »Terrorform«, aber auch andere Werkzeuge werden ähnlich gesehen.)

Was dieses Buch ist und was es nicht ist

Die These dieses Buchs ist, dass uns das Erforschen verschiedener Wege beim Einsatz von Werkzeugen zum Implementieren von Infrastruktur dabei helfen kann, die Qualität der von uns bereitgestellten Services zu verbessern. Wir wollen durch Geschwindigkeit und Auslieferungsfrequenz die Zuverlässigkeit und Qualität dessen, was wir ausliefern, verbessern.

Daher liegt der Fokus dieses Buchs weniger auf bestimmten Werkzeugen und mehr darauf, wie man sie einsetzt.

Auch wenn ich Beispiele für Tools für bestimmte Funktionen wie das Konfigurieren von Servern und das Provisionieren von Stacks erwähne, werden Sie keine Details zum Einsatz spezifischer Werkzeuge oder Cloud-Plattformen finden. Es gibt Patterns, Praktiken und Techniken, die unabhängig von den von Ihnen eingesetzten Tools und Plattformen relevant sein sollten.

Sie werden auch keine Codebeispiele für reale Werkzeuge oder Clouds finden. Tools ändern sich in diesem Bereich zu schnell, sodass Codebeispiele nicht aktuell gehalten werden könnten, aber die Ratschläge in diesem Buch sollten langsamer veralten und für viele Werkzeuge anwendbar sein. Stattdessen schreibe ich Pseudocode-Beispiele für fiktive Tools, um Konzepte deutlich zu machen. Auf der das

Buch begleitenden Website (<https://infrastructure-as-code.com>) finden Sie Beispielprojekte und -code.

Dieses Buch erklärt Ihnen nicht, wie Sie das Betriebssystem Linux einsetzen, Kubernetes-Cluster konfigurieren oder Networking-Routing betreiben. Aber es beschreibt Wege zum Provisionieren von Infrastruktur-Ressourcen, um diese Aufgaben umzusetzen, und wie Sie Code einsetzen, um sie auszuliefern. Ich stelle verschiedene Cluster-Topologie-Patterns und Ansätze zum Definieren und Managen von Clustern als Code vor. Ich beschreibe Patterns zum Provisionieren, Konfigurieren und Ändern von Server-Instanzen mithilfe von Code.

Sie sollten die Praktiken in diesem Buch durch Ressourcen zu den spezifischen Betriebssystemen, Clustering-Technologien und Cloud-Plattformen ergänzen. Auch hier erläutert dieses Buch Vorgehensweisen für den Einsatz dieser Werkzeuge und Technologien, die unabhängig vom spezifischen Tool relevant sind.

Dieses Buch streift Operability-Themen wie Monitoring und Observability, Log-Aggregation, Identity Management und andere Aspekte, die Sie zum Unterstützen von Services in einer Cloud-Umgebung benötigen, nur am Rande. Was Sie hier finden, soll Ihnen dabei helfen, die Infrastruktur als Code zu managen, die für diese Services erforderlich ist – die Details der spezifischen Services sind wiederum nur etwas, das Sie in entsprechenden Quellen finden.

Etwas Geschichte zu Infrastructure as Code

Tools und Praktiken rund um Infrastructure as Code gab es schon lange, bevor dieser Begriff geprägt wurde. Systemadministratorinnen und -administratoren nutzten schon von Anfang an Skripte, um Systeme zu managen. Mark Burgess hat das wegweisende CFEngine-System (<https://cfengine.com>) im Jahr 1993 geschaffen. Ich habe Praktiken zum Verwenden von Code zum vollständig automatisierten Provisionieren und Updaten von Servern in den frühen 2000ern über die Website <http://www.infrastructures.org> kennengelernt.¹

Infrastructure as Code ist zusammen mit der DevOps-Bewegung gewachsen. Andrew Clay-Shafer und Patrick Debois starteten die DevOps-Bewegung durch einen Talk auf der Agile-2008-Konferenz (<https://oreil.ly/ermR3>). Die erste Verwendung von »Infrastructure as Code« fand ich in einem Talk mit dem Titel »Agile Infrastructure« von Clay-Shafer auf der Velocity-Konferenz im Jahr 2009 (<https://oreil.ly/qnJKX>), und John Willis schrieb einen Artikel (https://oreil.ly/2F6y_), der diesen zusammenfasste. Adam Jacob, der Chef mitbegründet hat, und Luke Kanies, Begründer von Puppet, nutzten diese Phrase ebenfalls zu dieser Zeit.

¹ Es gibt auch im Sommer 2020 immer noch den ursprünglichen Inhalt auf dieser Site, allerdings hat sich seit 2007 nichts mehr darauf getan.

Für wen dieses Buch gedacht ist

Dieses Buch ist für Menschen gedacht, die mit dem Bereitstellen und Einsetzen von Infrastruktur zu tun haben, auf der Software ausgeliefert und ausgeführt wird. Sie haben vielleicht Erfahrung mit Systemen und Infrastruktur oder mit der Softwareentwicklung und -auslieferung. Ihre Rolle kann in der Entwicklung, dem Testen, der Architektur oder dem Management liegen. Ich gehe davon aus, dass Sie schon mit Cloud- oder virtualisierter Infrastruktur und Tools zum Automatisieren von Infrastructure as Code Kontakt hatten.

Leserinnen und Leser, für die Infrastructure as Code neu ist, sollten mit diesem Buch eine gute Einführung zum Thema erhalten, auch wenn Sie mehr daraus ziehen können, wenn Sie schon damit vertraut sind, wie Infrastruktur-Cloud-Plattformen arbeiten, und wenn Sie die Grundlagen zumindest eines Infrastruktur-Coding-Tools kennen.

Wenn Sie mehr Erfahrung im Umgang mit diesen Tools haben, finden Sie hier eine Mischung aus vertrauten und neuen Konzepten und Ansätzen. Der Inhalt sollte eine gemeinsame Sprache schaffen und Herausforderungen und Lösungen so beschreiben, dass erfahrene Praktiker und Teams Nutzen daraus ziehen können.

Prinzipien, Praktiken und Patterns

Ich verwende die Begriffe *Prinzipien*, *Praktiken* und *Patterns* (und *Antipatterns*), um damit zentrale Konzepte zu beschreiben. So setze ich sie ein:

Prinzip

Ein Prinzip ist eine Regel, die Ihnen dabei hilft, zwischen möglichen Lösungen auszuwählen.

Praktik

Eine Praktik ist eine Vorgehensweise, wie etwas umgesetzt wird. Eine gegebene Praktik ist nicht immer der einzige Weg, um etwas zu erledigen, und eventuell ist sie nicht einmal der beste Weg in einer bestimmten Situation. Sie sollten Prinzipien nutzen, um sich bei der Wahl der passendsten Praktik für eine gegebene Situation leiten zu lassen.

Pattern

Ein Pattern ist eine mögliche Lösung für ein Problem. Es ähnelt einer Praktik darin, dass andere Patterns in anderen Kontexten effektiver sein können. Jedes Pattern ist in einer Form beschrieben, die Ihnen dabei helfen soll, herauszufinden, wie relevant es für Ihr Problem ist.

Antipattern

Ein Antipattern kann zwar eine mögliche Lösung sein, Sie sollten es aber in den meisten Situationen nicht einsetzen, denn oft klingt es nur nach einer guten Idee – oder Sie geraten durch die Verwendung von Antipatterns in Situationen, die schwierig werden können, ohne dass Sie es bemerken.

Warum ich den Begriff »Best Practice« nicht verwende

Die Menschen in unserer Branche lieben es, über »Best Practices« zu sprechen. Problematisch an diesem Begriff ist, dass er uns oft dazu bringt, zu denken, dass es unabhängig von der Situation nur eine Lösung für ein Problem gibt.

Ich ziehe es vor, Praktiken und Patterns zu beschreiben und dabei darauf hinzuweisen, wann sie nützlich sind und welche Grenzen sie haben. Manches davon beschreibe ich als effektiver oder passender, aber ich versuche, offen für Alternativen zu sein. Bei Praktiken, die meiner Ansicht nach weniger effektiv sind, hoffe ich, dass ich meine Einschätzung nachvollziehbar erkläre.

Die ShopSpinner-Beispiele

Ich verwende in diesem Buch eine fiktive Firma namens ShopSpinner, um Konzepte zu illustrieren. Sie erstellt und betreibt Online-Stores für ihre Kunden.

ShopSpinner läuft auf FCS, dem Fictional Cloud Service – einem Public-IaaS-Provider mit verschiedenen Services, zu denen FSI (Fictional Server Images) und FKS (Fictional Kubernetes Service) gehören. Er verwendet das *Stackmaker*-Tool – ein Analogon zu Terraform, CloudFormation und Pulumi –, um Infrastruktur in seiner Cloud zu definieren und zu verwalten. FCS konfiguriert Server mit dem *Servermaker*-Tool, was für Tools wie Ansible, Chef oder Puppet steht.

Die Infrastruktur und das Systemdesign von ShopSpinner können je nachdem, was ich zu erklären versuche, unterschiedlich aussehen, wie auch die Code-Syntax oder Befehlszeilenargumente für die fiktiven Tools.

In diesem Buch verwendete Konventionen

Die folgenden typografischen Konventionen werden in diesem Buch genutzt:

Kursiv

Für neue Begriffe, URLs, E-Mail-Adressen, Dateinamen und Dateierweiterungen.

Schreibmaschinenschrift

Für Programmlistings, aber auch für Code-Fragmente in Absätzen, wie zum Beispiel Variablen- oder Funktionsnamen, Datenbanken, Datentypen, Umgebungsvariablen, Anweisungen und Schlüsselwörter.

###fette Schreibmaschinenschrift###

Für Befehle und anderen Text, der genau so vom Benutzer eingegeben werden sollte.

###kursive Schreibmaschinenschrift###

Für Text, der vom Benutzer durch eigene Werte ersetzt werden sollte.

**Tipp**

Dieses Symbol steht für einen Tipp oder Vorschlag.

**Hinweis**

Dieses Symbol steht für eine allgemeine Anmerkung.

**Warnung**

Dieses Symbol steht für eine Warnung oder Vorsichtsmaßnahme.

Danksagung

Wie schon die erste Auflage ist auch die vorliegende aktuelle 2. Auflage nicht alleine mein Werk – es ist das Ergebnis des möglichst guten Sammelns und Zusammenführens von Informationen, die ich von mehr Menschen gelernt habe, als ich mich erinnern, geschweige denn anständig aufzählen könnte. Eine große Entschuldigung und vielen Dank an all die, die ich hier beim Aufzählen vergessen habe.

Ich liebe es immer, zusammen mit James Lewis Ideen zu jonglieren – unsere Gespräche und seine Texte und Vorträge haben direkt und indirekt viel von dem beeinflusst, was Sie in diesem Buch finden. Er hat freundlicherweise seine umfassende Erfahrung zum Softwaredesign und sein Wissen über viele andere Themen mit mir geteilt, indem er mir seine Gedanken zu einer nahezu finalen Version dieses Buchs zukommen ließ. Seine Vorschläge haben mir dabei geholfen, die Verbindungen zu stärken, die ich zwischen der Softwareentwicklung und Infrastructure as Code ziehen wollte.

Martin Fowler hat meine Arbeit von Anfang an unterstützt. Seine Fähigkeit, aus der Erfahrung anderer zu lernen, sein Wissen und seine Ansichten anzuwenden und all das in klare, hilfreiche Ratschläge zu formulieren, inspiriert mich.

Thierry de Pauw war ein außerordentlich mitdenkender und hilfreicher Gutachter. Er hat mehrere Entwürfe des Buchs gelesen und seine Gedanken dazu mit mir geteilt, mir erzählt, was er neu und nützlich fand, welche Ideen mit seinen Erfahrungen übereinstimmen und welche Teile ihm nicht klar genug waren.

Ich danke Abigail Bangser, Jon Barber, Max Griffiths, Anne Simmons und Claire Walkley für ihre Unterstützung und Inspiration.

Mir haben verschiedene Personen, mit denen ich zusammengearbeitet habe, Gedanken und Ideen mitgeteilt, die dieses Buch besser gemacht haben. James Green hat mir vom Data Engineering und maschinellem Lernen im Kontext von Infrastruktur erzählt. Pat Downey hat seinen Einsatz von Expand and Contract für In-

Infrastruktur erklärt. Vincenzo Fabrizi hat mich auf den Wert der Inversion of Control für Infrastruktur-Abhängigkeiten hingewiesen. Effy Elden besitzt unfassbar viel Wissen über die diversen Infrastruktur-Tools. Moritz Heiber hat direkt und indirekt Einfluss auf dieses Buch genommen, auch wenn ich wohl nicht hoffen kann, dass er allem zu 100 Prozent zustimmt.

Bei ThoughtWorks habe ich die Möglichkeit, mit vielen Kolleginnen und Kollegen sowie Kundinnen und Kunden in Workshops, Projekten und Onlineforen über Infrastructure as Code zu sprechen. Zu ihnen gehören unter anderem Ama Asare, Nilakhya Chatterjee, Audrey Conceicao, Patrick Dale, Dhaval Doshi, Filip Fafara, Adam Fahie, John Feminella, Mario Fernandez, Louise Franklin, Heiko Gerin, Jarrod »Barry« Goodwin, Emily Gorcenski, James Gregory, Col Harris, Prince M Jain, Andrew Jones, Aiko Klostermann, Charles Korn, Vishwas Kumar, Punit Lad, Suya Liu, Tom Clement Oketch, Gerald Schmidt, Boss Supanat Pothivarakorn, Rodrigo Rech, Florian Sellmayr, Vladimir Sneblic, Isha Soni, Widyasari Stella, Paul Valla, Srikanth Venugopalan, Ankit Wal, Paul Yeoh und Jiayu Yi. Vielen Dank auch an Kent Spillner – dieses Mal weiß ich auch, warum.

Viele Personen haben unterschiedliche Entwurfsstände dieser Auflage gelesen und Feedback gegeben, unter anderem Artashes Arabajyan, Albert Attard, Simon Bisson, Phillip Campbell, Mario Cecchi, Carlos Conde, Bamdad Dashtban, Marc Hoffer, Willem van Ketwich, Barry O'Reilly, Rob Park, Robert Quinlivan, Wasin Wathanasirong und Rebecca Wirfs-Brock.

Großer Dank gebührt meiner Lektorin Virginia Wilson, die mich durch den langen und aufreibenden Prozess begleitet hat, durch den dieses Buch entstehen konnte. Mein Kollege John Amalanathan hat meine nur skizzierten Diagramme mit viel Geduld und Sorgfalt in die kleinen Kunstwerke verwandelt, die Sie in diesem Buch finden.

Mein Arbeitgeber ThoughtWorks hat mir ebenfalls sehr viel Unterstützung zukommen lassen. Vor allem, indem er die Umgebung geschaffen hat, in der ich von fantastischen Menschen lernen kann, aber auch durch das Fördern einer Kultur, die die Mitarbeiterinnen und Mitarbeiter darin unterstützt, Ideen mit der ganzen Branche zu teilen. Und schließlich kann ich bei ihm mit anderen ThoughtWorkern sowie Kundinnen und Kunden neue Wege der Arbeit erforschen und ausprobieren. Ashok Subramanian, Ruth Harrison, Renee Hawkins, Ken Mugrage, Rebecca Parsons und Gayathri Rao haben mir (neben vielen anderen) dabei geholfen, aus diesem Buch mehr als ein privates Projekt zu machen.

Und schließlich möchte ich Ozlem und Erel, die wieder einmal meine Obsession für dieses Buch erduldet haben, meiner ewigen Liebe versichern.

TEIL I

Grundlagen

