

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

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

Inhalt

Vorwort

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

Teil I Grundlagen

1 Was ist Infrastructure as Code?

Aus der Eisenzeit in das Cloud-Zeitalter
Infrastructure as Code
Vorteile von Infrastructure as Code
Infrastructure as Code nutzen, um für Änderungen zu optimieren
Einwand »Wir haben gar nicht so häufig Änderungen, sodass sich Automation nicht lohnt«
Einwand »Wir sollten erst bauen und danach automatisieren«

Einwand »Wir müssen zwischen Geschwindigkeit und Qualität entscheiden«

Die Four Key Metrics

Drei zentrale Praktiken für Infrastructure as Code

Zentrale Praktik: Definieren Sie alles als Code

Zentrale Praktik: Kontinuierliches Testen und die gesamte aktuelle Arbeit ausliefern

Zentrale Praktik: Kleine einfache Elemente bauen, die Sie unabhängig voneinander ändern können

Zusammenfassung

2 Prinzipien der Infrastruktur im Cloud-Zeitalter

Prinzip: Gehen Sie davon aus, dass Systeme unzuverlässig sind

Prinzip: Alles reproduzierbar machen

Fallstrick: Snowflake-Systeme

Prinzip: Erstellen Sie wegwerfbare Elemente

Prinzip: Variationen minimieren

Konfigurationsdrift

Prinzip: Stellen Sie sicher, dass Sie jeden Prozess wiederholen können

Zusammenfassung

3 Infrastruktur-Plattformen

Die Elemente eines Infrastruktur-Systems

Dynamische Infrastruktur-Plattform

Infrastruktur-Ressourcen

Computing-Ressourcen

Storage-Ressourcen

Networking-Ressourcen

Zusammenfassung

4 Zentrale Praktik: Definieren Sie alles als Code

Warum Sie Ihre Infrastruktur als Code definieren sollten

Was Sie als Code definieren können

- Wählen Sie Werkzeuge mit externalisierter Konfiguration aus
- Managen Sie Ihren Code in einer Versionsverwaltung
- Programmiersprachen für Infrastruktur
 - Infrastruktur-Scripting
 - Deklarative Infrastruktur-Sprachen
 - Programmierbare, imperative Infrastruktur-Sprachen
 - Deklarative und imperative Sprachen für Infrastruktur
 - Domänenspezifische Infrastruktur-Sprachen
 - Allgemein nutzbare Sprachen und DSLs für die Infrastruktur
- Implementierungs-Prinzipien beim Definieren von Infrastructure as Code
 - Halten Sie deklarativen und imperativen Code voneinander getrennt
 - Behandeln Sie Infrastruktur-Code wie echten Code
- Zusammenfassung

Teil II Arbeiten mit Infrastruktur-Stacks

5 Infrastruktur-Stacks als Code bauen

- Was ist ein Infrastruktur-Stack?
 - Stack-Code
 - Stack-Instanzen
 - Server in einem Stack konfigurieren
 - Low-Level-Infrastruktur-Sprachen
 - High-Level-Infrastruktur-Sprachen
- Patterns und Antipatterns für das Strukturieren von Stacks
 - Antipattern: Monolithic Stack
 - Pattern: Application Group Stack
 - Pattern: Service Stack

Pattern: Micro Stack
Zusammenfassung

6 Umgebungen mit Stacks bauen

Worum es bei Umgebungen geht

Auslieferungsumgebungen

Mehrere Produktivumgebungen

Umgebungen, Konsistenz und Konfiguration

Patterns zum Bauen von Umgebungen

Antipattern: Multiple-Environment Stack

Antipattern: Copy-Paste Environments

Pattern: Reusable Stack

Umgebungen mit mehreren Stacks erstellen

Zusammenfassung

7 Stack-Instanzen konfigurieren

Eindeutige Kennungen durch Stack-Parameter erstellen

Beispiel-Stack-Parameter

Patterns zum Konfigurieren von Stacks

Antipattern: Manual Stack Parameters

Pattern: Stack Environment Variables

Pattern: Scripted Parameters

Pattern: Stack Configuration Files

Pattern: Wrapper-Stack

Pattern: Pipeline Stack Parameters

Pattern: Stack Parameter Registry

Konfigurations-Registry

Eine Konfigurations-Registry implementieren

Eine oder mehrere Konfigurations-Registries

Secrets als Parameter nutzen

Secrets verschlüsseln

Secretlose Autorisierung

Secrets zur Laufzeit injizieren

Wegwerf-Secrets

Zusammenfassung

8 Zentrale Praktik: Kontinuierlich testen und ausliefern

Warum Infrastruktur-Code kontinuierlich testen?

Was kontinuierliches Testen bedeutet

Was sollten wir bei der Infrastruktur testen?

Herausforderungen beim Testen von Infrastruktur-Code

Herausforderung: Tests für deklarativen Code haben häufig nur einen geringen Wert

Herausforderung: Das Testen von Infrastruktur-Code ist langsam

Herausforderung: Abhängigkeiten verkomplizieren die Test-Infrastruktur

Progressives Testen

Testpyramide

Schweizer-Käse-Testmodell

Infrastruktur-Delivery-Pipelines

Pipeline-Stages

Scope von Komponenten, die in einer Stage getestet werden

Scope von Abhängigkeiten für eine Stage

Plattformelemente, die für eine Stage erforderlich sind

Software und Services für die Delivery-Pipeline

Testen in der Produktivumgebung

Was Sie außerhalb der Produktivumgebung nicht nachbauen können

Die Risiken beim Testen in der Produktivumgebung managen

Zusammenfassung

9 Infrastruktur-Stacks testen

Beispiel-Infrastruktur

Der Beispiel-Stack

Pipeline für den Beispiel-Stack

Offline-Test-Stages für Stacks

Syntax-Checks

- Statische Offline-Code-Analyse
- Statische Code-Analyse per API
- Testen mit einer Mock-API
- Online-Test-Stages für Stacks
 - Preview: Prüfen, welche Änderungen vorgenommen werden
 - Verifikation: Aussagen über Infrastruktur-Ressourcen treffen
 - Ergebnisse: Prüfen, dass die Infrastruktur korrekt arbeitet
- Test-Fixtures für den Umgang mit Abhängigkeiten verwenden
 - Test-Doubles für Upstream-Abhängigkeiten
 - Test-Fixtures für Downstream-Abhängigkeiten
 - Komponenten refaktorisieren, um sie isolieren zu können
- Lebenszyklus-Patterns für Testinstanzen von Stacks
 - Pattern: Persistent Test Stack
 - Pattern: Ephemeral Test Stack
 - Antipattern: Dual Persistent and Ephemeral Stack Stages
 - Pattern: Periodic Stack Rebuild
 - Pattern: Continuous Stack Reset
- Test-Orchestrierung
 - Unterstützen Sie lokales Testen
 - Vermeiden Sie eine enge Kopplung mit Pipeline-Tools
 - Tools zur Test-Orchestrierung
- Zusammenfassung

Teil III Mit Servern und anderen Anwendungs-Laufzeitplattformen arbeiten

10 Anwendungs-Laufzeitumgebungen

Cloud-native und anwendungsgesteuerte Infrastruktur

- Ziele für eine Anwendungs-Laufzeitumgebung
 - Deploybare Teile einer Anwendung
 - Deployment-Pakete
- Anwendungen auf Server deployen
 - Anwendungen als Container verpacken
 - Anwendungen auf Server-Cluster deployen
- Anwendungen auf Anwendungs-Cluster deployen
- Pakete zum Deployen von Anwendungen auf Cluster
- FaaS-Serverless-Anwendungen deployen
- Anwendungsdaten
 - Datenschemata und -strukturen
 - Cloud-native Storage-Infrastruktur für Anwendungen
- Anwendungs-Connectivity
- Service Discovery
- Zusammenfassung

11 Server als Code bauen

- Was gibt es auf einem Server
- Woher Dinge kommen
- Server-Konfigurationscode
 - Code-Module für die Serverkonfiguration
 - Code-Module für die Serverkonfiguration designen
 - Server-Code versionieren und weitergeben
 - Serverrollen
- Server-Code testen
 - Server-Code progressiv testen
 - Was Sie bei Server-Code testen
 - Wie Sie Server-Code testen
- Eine neue Server-Instanz erstellen
 - Eine neue Server-Instanz per Hand erstellen
 - Einen Server mit einem Skript erstellen
 - Einen Server mit einem Stack-Management-Tool erstellen
 - Die Plattform für das automatische Erstellen von Servern konfigurieren

Einen Server mit einem Network-Provisioning-Tool erstellen

Server vorbereiten

Hot-Cloning eines Servers

Einen Server-Snapshot verwenden

Ein sauberes Server-Image erstellen

Eine neue Server-Instanz konfigurieren

Eine Server-Instanz ausbacken

Server-Images backen

Backen und Ausbacken kombinieren

Serverkonfiguration beim Erstellen eines Servers anwenden

Zusammenfassung

12 Änderungen an Servern managen

Patterns zum Changemanagement: Wann Änderungen angewendet werden

Antipattern: Apply on Change

Pattern: Continuous Configuration Synchronization

Pattern: Immutable Server

Wie Sie Serverkonfigurationscode anwenden

Pattern: Push Server Configuration

Pattern: Pull Server Configuration

Andere Ereignisse im Lebenszyklus eines Servers

Eine Server-Instanz stoppen und erneut starten

Eine Server-Instanz ersetzen

Einen ausgefallenen Server wiederherstellen

Zusammenfassung

13 Server-Images als Code

Ein Server-Image bauen

Warum ein Server-Image bauen?

Wie Sie ein Server-Image bauen

Tools zum Bauen von Server-Images

Online Image Building

Offline Image Building

- Ursprungsinhalte für ein Server-Image
 - Aus einem Stock-Server-Image bauen
 - Ein Server-Image von Grund auf bauen
 - Herkunft eines Server-Image und seiner Inhalte
- Ein Server-Image ändern
 - Ein frisches Image aufwärmen oder backen
 - Ein Server-Image versionieren
 - Server-Instanzen aktualisieren, wenn sich ein Image ändert
 - Ein Server-Image in mehreren Teams verwenden
 - Umgang mit größeren Änderungen an einem Image
- Eine Pipeline zum Testen und Ausliefern eines Server-Image verwenden
 - Build-Stage für ein Server-Image
 - Test-Stage für ein Server-Image
 - Delivery-Stages für ein Server-Image
- Mehrere Server-Images verwenden
 - Server-Images für unterschiedliche Infrastruktur-Plattformen
 - Server-Images für unterschiedliche Betriebssysteme
 - Server-Images für unterschiedliche Hardware-Architekturen
 - Server-Images für unterschiedliche Rollen
 - Server-Images in Schichten erstellen
 - Code für mehrere Server-Images verwenden
- Zusammenfassung

14 Cluster als Code bauen

- Lösungen für Anwendungs-Cluster
 - Cluster as a Service
 - Packaged Cluster Distribution
- Stack-Topologien für Anwendungs-Cluster
 - Monolithischer Stack, der Cluster as a Service nutzt
 - Monolithischer Stack für eine Packaged-Cluster-Lösung

- Pipeline für einen monolithischen Anwendungs-Cluster-Stack
- Beispiel für mehrere Stacks in einem Cluster
- Strategien zur gemeinsamen Verwendung von Anwendungs-Clustern
 - Ein großes Cluster für alles
 - Getrennte Cluster für Auslieferungs-Stages
 - Cluster für die Governance
 - Cluster für Teams
 - Service Mesh
- Infrastruktur für FaaS Serverless
- Zusammenfassung

Teil IV Infrastruktur designen

15 Zentrale Praktik: Kleine, einfache Elemente

- Für Modularität designen
 - Eigenschaften gut designer Komponenten
 - Regeln für das Designen von Komponenten
 - Design-Entscheidungen durch Testen
- Infrastruktur modularisieren
 - Stack-Komponenten versus Stacks als Komponenten
 - Einen Server in einem Stack verwenden
- Grenzen zwischen Komponenten ziehen
 - Grenzen mit natürlichen Änderungsmustern abstimmen
 - Grenzen mit Komponenten-Lebenszyklen abstimmen
 - Grenzen mit Organisationsstrukturen abstimmen
 - Grenzen schaffen, die Resilienz fördern
 - Grenzen schaffen, die Skalierbarkeit ermöglichen
 - Grenzen auf Sicherheits- und Governance-Aspekte abstimmen
- Zusammenfassung

16 Stacks aus Komponenten bauen

- Infrastruktur-Sprachen für Stack-Komponenten
 - Deklarativen Code mit Modulen wiederverwenden
 - Stack-Elemente dynamisch mit Bibliotheken erstellen

- Patterns für Stack-Komponenten
 - Pattern: Facade Module
 - Antipattern: Obfuscation Module
 - Antipattern: Unshared Module
 - Pattern: Bundle Module
 - Antipattern: Spaghetti Module
 - Pattern: Infrastructure Domain Entity
- Eine Abstraktionsschicht bauen
- Zusammenfassung

17 Stacks als Komponenten einsetzen

- Abhängigkeiten zwischen Stacks erkennen
 - Pattern: Resource Matching
 - Pattern: Stack Data Lookup
 - Pattern: Integration Registry Lookup
 - Dependency Injection
- Zusammenfassung

Teil V Infrastruktur bereitstellen

18 Infrastruktur-Code organisieren

- Projekte und Repositories organisieren
 - Ein Repository oder viele?
 - Ein Repository für alles
 - Ein eigenes Repository für jedes Projekt (Microrepo)
 - Mehrere Repositories mit mehreren Projekten
- Unterschiedliche Arten von Code organisieren
 - Projektsupport-Dateien
 - Projektübergreifende Tests
 - Dedizierte Projekte für Integrationstests
 - Code anhand des Domänenkonzepts organisieren

- Dateien mit Konfigurationswerten organisieren
- Infrastruktur- und Anwendungscode managen
- Infrastruktur und Anwendungen ausliefern
- Anwendungen mit Infrastruktur testen
- Infrastruktur vor der Integration testen
- Infrastruktur-Code zum Deployen von Anwendungen nutzen

Zusammenfassung

19 Infrastruktur-Code ausliefern

- Auslieferungsprozess von Infrastruktur-Code
 - Ein Infrastruktur-Projekt bauen
 - Infrastruktur-Code als Artefakt verpacken
 - Infrastruktur-Code mit einem Repository ausliefern
- Projekte integrieren
 - Pattern: Build-Time Project Integration
 - Pattern: Delivery-Time Project Integration
 - Pattern: Apply-Time Project Integration
- Infrastruktur-Tools durch Skripte verpacken
 - Konfigurationswerte zusammenführen
 - Wrapper-Skripte vereinfachen

Zusammenfassung

20 Team-Workflows

- Die Menschen
- Wer schreibt Infrastruktur-Code?
- Code auf Infrastruktur anwenden
 - Code von Ihrem lokalen Rechner aus anwenden
 - Code von einem zentralisierten Service anwenden lassen
 - Private Infrastruktur-Instanzen
 - Quellcode-Banches in Workflows
- Konfigurationsdrift verhindern
 - Automatisierungs-Verzögerung minimieren
 - Ad-hoc-Anwendung vermeiden
 - Code kontinuierlich anwenden

Immutable Infrastruktur
Governance in einem Pipeline-basierten Workflow
Zuständigkeiten neu ordnen
Shift Left
Ein Beispielprozess für Infrastructure as Code mit
Governance
Zusammenfassung

21 Infrastruktur sicher ändern

Reduzieren Sie den Umfang von Änderungen
Kleine Änderungen
Refaktorisieren - ein Beispiel
Unvollständige Änderungen in die Produktivumgebung
übernehmen
Parallele Instanzen
Abwärtskompatible Transformationen
Feature Toggles
Live-Infrastruktur ändern
Infrastruktur-Chirurgie
Expand and Contract
Zero-Downtime-Änderungen
Kontinuität
Kontinuität durch das Verhindern von Fehlern
Kontinuität durch schnelles Wiederherstellen
Kontinuierliches Disaster Recovery
Chaos Engineering
Für Ausfälle planen
Datenkontinuität in einem sich ändernden System
Sperren
Aufteilen
Replizieren
Neu laden
Ansätze zur Datenkontinuität mischen
Zusammenfassung

Index

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

Vorwort

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

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-Objekte 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.

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