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]

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 Codebeïspiele 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-

frastruktur 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, Jarrad »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 Hofer, Willem van Ketwich, Barry O'Reilly, Rob Park, Robert Quinlivan, Wasin Watthanasrisong 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.