KAPITEL 1

Was ist Infrastructure as Code?

Arbeiten Sie in einem Team, das IT-Infrastruktur erstellt und betreibt, sollten Ihnen Automatisierungstechniken für Cloud und Infrastruktur dabei helfen, in weniger Zeit zuverlässiger mehr Wert bieten zu können. Aber in der Praxis geht es vor allem darum, die stetig zunehmende Größe, Komplexität und Diversität der verschiedenen Elemente im Griff zu behalten.

Diese Technologien sind besonders relevant, wenn Organisationen digital werden. Wenn Business-Menschen »Digital« sagen, meinen sie damit, dass Softwaresysteme für das, was die Organisation tut, von zentraler Bedeutung sind.[1] Der Wechsel ins Digitale verstärkt den Druck auf Sie, mehr Aufgaben schneller zu erledigen. Sie müssen zusätzliche Services hinzufügen und betreuen. Mehr Business-Aktivitäten. Mehr Mitarbeiterinnen und Mitarbeiter. Mehr Kundinnen und Kunden, Lieferpartner und andere Stakeholder.

Cloud- und Automatisierungs-Tools helfen dabei, es viel leichter zu machen, Infrastruktur hinzuzufügen und anzupassen. Aber viele Teams haben Probleme damit, ausreichend Zeit dafür zu finden, mit der schon vorhandenen Infrastruktur Schritt zu halten. Da ist es nicht sehr hilfreich, das Erstellen von noch mehr Infrastruktur einfacher zu machen. Einer meiner Kunden sagte mir: »Durch Cloud wurden die Barrieren eingerissen, die unseren Reifenbrand unter Kontrolle behielten.«[2]

Viele haben auf die Drohung eines ausufernden Chaos durch ein Anziehen ihres Änderungsmanagement-Prozesses reagiert. Sie haben die Hoffnung, dass sich das Chaos durch Begrenzen und Kontrollieren der Änderungen vermeiden lässt. Also legen sie die Cloud in Ketten.

1 Das steht im Gegensatz zu dem, was die gleichen Personen ein paar Jahre zuvor gesagt haben, nämlich dass Software »nicht Teil unseres Kerngeschäfts ist«. Nachdem sie diesem Ratschlag gefolgt sind und ihre IT outgesourced haben, stellten die Organisationen fest, dass diejenigen an ihnen vorbeizogen, die bessere Software als Wettbewerbsvorteil und nicht als Kostenersparnis ansahen.

2 Laut (englischsprachiger) Wikipedia (*https://oreil.ly/IkDu_*) kann es einen *Reifenbrand* in zwei Ausprägungen geben: »Schnell brennende Ereignisse, die fast sofort dazu führen, dass das Feuer außer Kontrolle gerät, und langsam brennende Pyrolyse, die sich mehr als ein Jahrzehnt hinziehen kann.«

Damit gibt es zwei Probleme. Das eine ist, dass so die Vorteile der Cloud-Technologien verloren gehen, das andere, dass die Benutzerinnen und Benutzer die Vorteile der Cloud-Technologie auch haben *wollen*. So ist man bemüht, die Personen zu umgehen, die versuchen, das Chaos im Griff zu behalten. Im schlimmsten Fall wird das Risikomanagement vollständig ignoriert, und man entscheidet für sich, dass das in der schönen neuen Cloud-Welt nicht notwendig sei. Sie setzen auf Cowboy-IT, was zu verschiedenen Problemen führt.[1]

Die Prämisse dieses Buchs ist, dass Sie Cloud- und Automatisierungs-Technologie nutzen können, um Änderungen einfach, sicher, schnell und verantwortungsbewusst durchführen zu können. Diese Vorteile entstehen nicht automatisch durch den Einsatz von Automatisierungs-Werkzeugen oder Cloud-Plattformen. Sie hängen davon ab, wie Sie diese Technologie verwenden.

DevOps und Infrastructure as Code

DevOps ist eine Bewegung, die Hindernisse abbaut und Reibungen zwischen der Silo-Entwicklung, Operations und anderen Stakeholdern reduziert, die am Planen, Bauen und Ausführen von Software beteiligt sind. Auch wenn der Technologie-Anteil der sichtbarste und in mancherlei Hinsicht auch einfachste Aspekt von DevOps ist, haben Kultur, Personen und Prozesse den größten Einfluss auf den Fluss und die Effektivität. Technologie und Entwicklungspraktiken wie Infrastructure as Code sollten zum Einsatz kommen, um die Aktivitäten zu unterstützen, die Gräben überbrücken und die Zusammenarbeit verbessern.

In diesem Kapitel erkläre ich, dass moderne, dynamische Infrastruktur eine Mentalität aus dem »Cloud-Zeitalter« erfordert. Die Mentalität unterscheidet sich grundlegend vom klassischen »Eisenzeit«-Ansatz, den wir bei statischen Prä-Cloud-Systemen genutzt haben. Ich definiere drei zentrale Praktiken für das Implementieren von Infrastructure as Code: Definiere alles als Code, teste und liefere alles fortlaufend während des Entwickelns aus und baue das System aus kleinen, lose gekoppelten Elementen auf.

Ich beschreibe zudem in diesem Kapitel die Gründe für den »Cloud-Zeitalter«-Ansatz zur Infrastruktur. Dieser löst sich von der fälschlich angenommenen Gegensätzlichkeit von Geschwindigkeit und Qualität, die man gegeneinander abwägen müsse. Stattdessen nutzen wir die Geschwindigkeit als eine Möglichkeit, die Qualität zu verbessern, und die Qualität, um mit hoher Geschwindigkeit auszuliefern.

1 Mit »Cowboy-IT« meine ich Leute, die IT-Systeme ohne eine Vorgehensweise oder Überlegung hinsichtlich der Konsequenzen in der Zukunft bauen. Meist wählen Personen, die noch nie Produktivsysteme betreut haben, den einfachsten Weg, um etwas zum Laufen zu bekommen, ohne auf Sicherheit, Wartbarkeit, Performance und andere Operability-Aspekte zu achten.

Aus der Eisenzeit in das Cloud-Zeitalter

Technologie des Cloud-Zeitalters ermöglicht ein schnelleres Provisionieren und Ändern von Infrastruktur, als dies mit den klassischen Technologien aus der Eisenzeit möglich wäre (Tabelle 1-1).

Tabelle 1-1: Technologische Änderungen im Cloud-Zeitalter

Eisenzeit	Cloud-Zeitalter
Physische Hardware	Virtualisierte Ressourcen
Provisionieren dauert Wochen	Provisionieren dauert Minuten
Manuelle Prozesse	Automatisierte Prozesse

Aber diese Technologien machen es nicht notwendigerweise einfacher, Ihre Systeme zu managen und wachsen zu lassen. Überführen Sie ein System mit technischen Schulden (*https://oreil.ly/3AqHB*) in eine schrankenlose Cloud-Infrastruktur, beschleunigen Sie nur das Chaos.

Vielleicht konnten Sie auf bewährte, klassische Governance-Modelle zurückgreifen, um die Geschwindigkeit und das Chaos zu kontrollieren, das neuere Technologien mit sich bringen. Ein umfassendes, vorher ausgearbeitetes Design, rigorose Change Reviews und strikt getrennte Zuständigkeiten werden schon für Ordnung sorgen!

Aber leider sind diese Modelle für die Eisenzeit optimiert, in der Änderungen langsam und teuer sind. Sie sorgen für zusätzlichen Aufwand im Vorhinein mit der Hoffnung, später den Zeitaufwand für die Änderung zu verringern. Das ist durchaus sinnvoll, wenn es später teuer und langsam ist, Änderungen vorzunehmen. Aber durch die Cloud werden Änderungen schnell und günstig. Sie sollten diese Geschwindigkeit zu Ihrem Vorteil nutzen, um kontinuierlich zu lernen und Ihr System zu verbessern. Arbeiten Sie weiter wie in der Eisenzeit, sind Ihr Lernen und Ihre Verbesserungen massiv eingeschränkt.

Statt also langsame Prozesse aus der Eisenzeit auf schnellere Technologie des Cloud-Zeitalters anzuwenden, sollten Sie eine neue Mentalität übernehmen. Nutzen Sie eine schnellere Technologie, um Risiken zu verringern und die Qualität zu verbessern. Das erfordert einen grundlegend anderen Ansatz und neue Wege, über Änderungen und Risiken nachzudenken (Tabelle 1-2).

Tabelle 1-2: Arbeitsweisen im Cloud-Zeitalter

Eisenzeit	Cloud-Zeitalter
Änderungskosten sind hoch	Änderungskosten sind niedrig
Änderungen stehen für Fehler (Änderungen müssen »gemanagt« oder »kontrolliert« werden)	Änderungen stehen für Lernen und Verbesserungen

Tabelle 1-2: Arbeitsweisen im Cloud-Zeitalter (Fortsetzung)

Eisenzeit	Cloud-Zeitalter
Gelegenheit zu Fehlern verringern	Geschwindigkeit von Verbesserungen maximieren
In großen Batches ausliefern, am Ende testen	Kleine Änderungen ausliefern, fortlaufend testen
Lange Release-Zyklen	Kurze Release-Zyklen
Monolithische Architektur (wenige und größere Komponenten)	Microservices-Architektur (viele und kleinere Komponenten)
Konfiguration über GUI oder direkt an der Hardware	Konfiguration als Code

Infrastructure as Code

Infrastructure as Code ist ein Ansatz für die Infrastruktur-Automatisierung, der auf Praktiken aus der Softwareentwicklung basiert. Er baut auf konsistenten, wiederholbaren Routinen zum Provisionieren und zur Änderung von Systemen und deren Konfiguration auf. Sie nehmen Änderungen am Code vor und nutzen dann Automation, um diese Änderungen zu testen und auf Ihre Systeme anzuwenden.

In diesem Buch erläutere ich, wie Sie agile Entwicklungspraktiken wie Test-driven Development (TDD), Continuous Integration (CI) und Continuous Delivery (CD) nutzen können, um das Ändern von Infrastruktur schnell und sicher zu gestalten. Ich beschreibe zudem, wie ein modernes Softwaredesign für resiliente, gut gewartete Infrastruktur sorgen kann. Diese Praktiken und Design-Ansätze unterstützen sich gegenseitig. Gut designte Infrastruktur lässt sich leichter testen und bereitstellen. Automatisiertes Testen und Bereitstellen führt zu einfacherem und sauberem Design.

Vorteile von Infrastructure as Code

Zusammengefasst kann man sagen, dass Organisationen, die Infrastructure as Code nutzen, um dynamische Infrastruktur zu verwalten, unter anderem auf Folgendes hoffen:

- Mit der IT-Infrastruktur ein schnelles Bereitstellen von Werten ermöglichen.
- Aufwand und Risiken von Änderungen an der Infrastruktur reduzieren.
- Anwenderinnen und Anwender der Infrastruktur dazu befähigen, die notwendigen Ressourcen dann zu bekommen, wenn sie sie brauchen.
- Gemeinsame Werkzeuge für Entwicklung, Operations und andere Stakeholder bereitstellen.
- Systeme aufbauen, die zuverlässig, sicher und kostengünstig sind.
- Steuerelemente für Governance, Sicherheit und Compliance sichtbar machen.
- Die Geschwindigkeit verbessern, mit der Fehler gefunden und gelöst werden.

Infrastructure as Code nutzen, um für Änderungen zu optimieren

Angesichts dessen, dass Änderungen das größte Risiko für ein Produktivsystem sind, sich kontinuierliche Änderungen nicht vermeiden lassen und ein System nur durch Änderungen verbesserbar ist, ist es sinnvoll, Ihre Fähigkeit zu optimieren, Änderungen sowohl schnell *als auch* zuverlässig zu machen.[1] Die Forschungsergebnisse im *Accelerated State of DevOps Report* unterstützen diese Ansicht. Häufige und zuverlässige Änderungen korrelieren mit dem Erfolg von Organisationen.[2]

Es gibt eine Reihe von Einwänden, die ich zu hören bekomme, wenn ich einem Team empfehle, zum Optimieren von Änderungen Automation einzuführen. Ich glaube, diese entstehen aus Missverständnissen, wie Sie Automation verwenden können und sollten.

Einwand »Wir haben gar nicht so häufig Änderungen, sodass sich Automation nicht lohnt«

Wir möchten gerne glauben, dass wir ein System bauen, und dann ist es »fertig«. Bei einer solchen Sichtweise nehmen wir nicht viele Änderungen vor, daher ist das Automatisieren von Änderungen nur Zeitverschwendung.

In der Realität gibt es nur an sehr wenigen Systemen keine Veränderungen mehr – zumindest nicht, bevor sie ausgemustert werden. Manche gehen davon aus, dass die aktuelle Menge an Änderungen nur vorübergehend ist. Andere schaffen aufwendige Prozesse zum Änderungsmanagement, um Personen davon abzuhalten, nach Änderungen zu fragen. Aber diejenigen reden es sich schön. Die meisten Teams, die aktiv genutzte Systeme betreuen, haben es mit einem kontinuierlichen Strom von Änderungen zu tun.

Denken Sie nur an die folgenden Beispiele für Änderungen an der Infrastruktur:

- Ein wichtiges neues Anwendungsfeature erfordert das Hinzufügen einer neuen Datenbank.
- Für ein neues Anwendungsfeature muss der Anwendungsserver aktualisiert werden.
- Die App wird häufiger genutzt als erwartet. Sie brauchen mehr Server, neue Cluster und zusätzliche Networking- und Storage-Kapazitäten.

1 Gene Kim, George Spafford und Kevin Behr schreiben in *The Visible Ops Handbook* (IT Process Institute), dass Änderungen für 80 Prozent der ungeplanten Ausfälle verantwortlich sind.

2 Die Berichte aus der *Accelerate*-Forschung finden sich im jährlich erscheinenden *State of DevOps Report* (*https://oreil.ly/0Q3FE*) und im Buch *Accelerate* von Dr. Nicole Forsgren, Jez Humble und Gene Kim (IT Revolution Press).

- Beim Performance-Profiling zeigt sich, dass die aktuelle Deployment-Architektur für die Anwendung die Performance einschränkt. Sie müssen die Anwendungen auf mehrere andere Anwendungsserver neu deployen. Dazu sind Änderungen am Clustering und an der Netzwerk-Architektur erforderlich.
- Es gibt eine neu bekannt gewordene Sicherheitslücke in Systempaketen für Ihr Betriebssystem. Sie müssen dutzende Produktivserver patchen.
- Sie müssen Server aktualisieren, auf denen veraltete Versionen des Betriebssystems und zentraler Pakete laufen.
- Auf Ihren Webservern gibt es immer wieder Aussetzer. Sie müssen eine Reihe von Konfigurationsänderungen vornehmen, um das Problem untersuchen zu können. Dann müssen Sie ein Modul aktualisieren, um das Problem zu lösen.
- Sie finden eine Konfigurationsänderung, die die Performance Ihrer Datenbank verbessert.

Eine zentrale Wahrheit des Cloud-Zeitalters ist: *Stabilität entsteht durch Änderungen.*

Ungepatchte Systeme sind nicht stabil – sie sind angreifbar. Können Sie Probleme nicht beheben, sobald Sie sie finden, ist Ihr System nicht stabil. Können Sie nach einem Ausfall nicht schnell wieder auf die Füße kommen, ist Ihr System nicht stabil. Wenn für Ihre Änderungen eine deutliche Downtime erforderlich ist, ist Ihr System nicht stabil. Wenn Änderungen immer wieder fehlschlagen, ist Ihr System nicht stabil.

Einwand »Wir sollten erst bauen und danach automatisieren«

Wenn Sie mit Infrastructure as Code beginnen, haben Sie eine steile Lernkurve vor sich. Das Aufsetzen der Tools, Services und Arbeitsweisen zum Automatisieren der Infrastruktur-Bereitstellung erfordert viel Arbeit, insbesondere, wenn Sie sich gleichzeitig auch noch in eine neue Infrastruktur-Plattform einarbeiten. Der Wert dieser Arbeit wird erst dann sichtbar, wenn Sie beginnen, Services damit zu bauen und zu deployen. Und auch dann wird er eventuell denen nicht deutlich werden, die nicht direkt mit der Infrastruktur arbeiten.

Stakeholder drängen Infrastruktur-Teams oft dazu, schnell und mal eben manuell neue in der Cloud gehostete Systeme zu bauen und sich erst später um das Automatisieren zu kümmern.

Es gibt drei Gründe dafür, warum das eine schlechte Idee ist:

- Durch die Automation sollte das Bereitstellen schneller gehen, auch für neue Elemente. Implementieren Sie die Automation erst, nachdem ein Großteil der Arbeit erledigt ist, opfern Sie viele der Vorteile.
- Automation erleichtert das Schreiben automatisierter Tests für das, was Sie bauen. Und sie macht es einfacher, etwas schnell zu korrigieren und neu zu bauen, wenn Sie Probleme finden. Wenn Sie dies als Teil des Build-Prozesses tun, hilft es Ihnen dabei, eine bessere Infrastruktur zu bauen.

- Das Automatisieren eines bestehenden Systems ist sehr schwer. Automation ist Teil des Designs und der Implementierung eines Systems. Um ein System, das ohne Automation aufgebaut wurde, um diese zu ergänzen, müssen Sie das Design und die Implementierung des Systems deutlich anpassen. Das gilt auch für das automatisierte Testen und Deployen.

Cloud-Infrastruktur, die ohne Automation aufgebaut wurde, müssen Sie schneller abschreiben, als Sie denken. Die Kosten für das manuelle Warten und Beheben von Fehlern im System können schnell deutlich wachsen. Ist der Service, der darauf läuft, erfolgreich, werden Sie die Stakeholder dazu drängen, ihn zu erweitern und neue Features hinzuzufügen, statt innezuhalten, um ihn neu zu bauen.

Das Gleiche gilt, wenn Sie ein System als Experiment bauen. Haben Sie einen Proof of Concept zum Laufen gebracht, wird es den Druck geben, sich mit dem nächsten Thema zu beschäftigen, statt zurückzukehren und das System richtig neu zu bauen. Und eigentlich sollte Automation auch Teil des Experiments sein. Wollen Sie Automation zum Managen Ihrer Infrastruktur einsetzen, müssen Sie verstehen, wie das funktionieren wird, daher sollte es auch Teil Ihres Proof of Concept sein.

Die Lösung besteht darin, Ihr System inkrementell aufzubauen und die Automation direkt mit zu berücksichtigen. Stellen Sie sicher, dass Sie Werte bieten, während Sie gleichzeitig für die Möglichkeit sorgen, das kontinuierlich zu tun.

Einwand »Wir müssen zwischen Geschwindigkeit und Qualität entscheiden«

Es ist ganz natürlich, davon auszugehen, dass Sie nur dann schnell vorankommen können, wenn Sie Abstriche bei der Qualität machen, und dass Sie nur dann Qualität erhalten, wenn Sie sich langsam bewegen. Sie sehen das vielleicht als Kontinuum (siehe Abbildung 1-1).

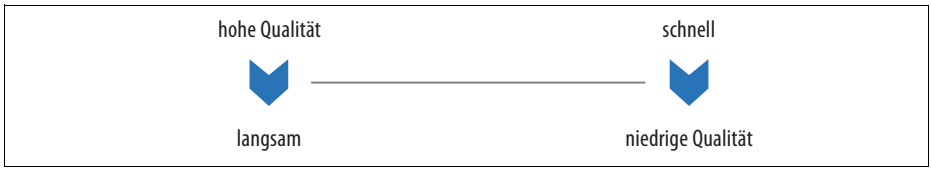

Abbildung 1-1: Die Idee, dass sich Geschwindigkeit und Qualität an entgegengesetzten Enden eines Spektrums befinden, ist eine fälschlich angenommene Gegensätzlichkeit.

Die Accelerate-Forschungsdaten, die ich weiter oben erwähnt habe (siehe »Infrastructure as Code nutzen, um für Änderungen zu optimieren« auf Seite 35), zeigen aber etwas anderes:

> Diese Ergebnisse zeigen, dass es keinen Kompromiss zwischen dem Verbessern der Performance und dem Erreichen einer besseren Stabilität und Qualität gibt. Stattdessen sind High-Performer bei all diesen Messwerten besser. Das ist genau das, was die Agile- und Lean-Bewegungen predigen, aber ein Glaubenssatz in unserer Branche basiert immer noch auf der falschen Annahme, dass man ein schnelles Vorankommen gegen andere Performance-Ziele abwägen muss, statt alles zu unterstützen und sich gegenseitig verstärken zu lassen.
>
> *– Nicole Forsgren, PhD, Accelerate*

Kurz gesagt können Organisationen nicht wählen, ob sie entweder Veränderungen oder Stabilität gut hinbekommen. Sie tendieren dazu, entweder in beidem gut oder in beidem schlecht zu sein.

Ich ziehe es vor, Qualität und Geschwindigkeit als Quadrant statt als Kontinuum zu betrachten,[1] wie in Abbildung 1-2 zu sehen ist.

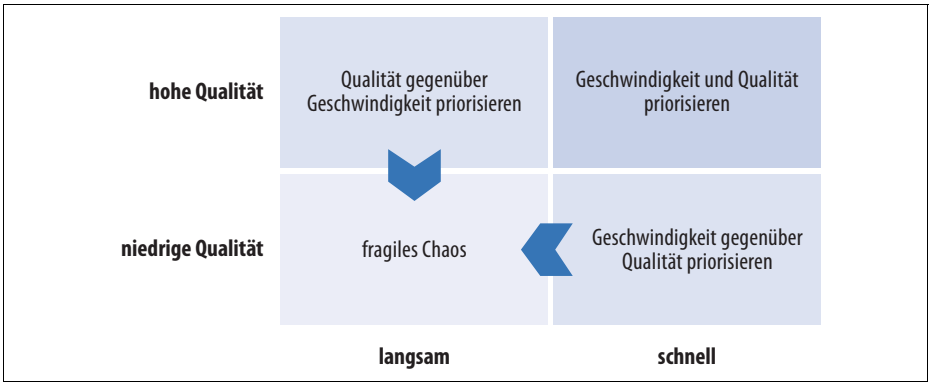

Abbildung 1-2: Geschwindigkeit und Qualität auf Quadranten abgebildet

Dieses Quadrantenmodell zeigt, warum es auf jeden Fall zu Mittelmäßigkeit führen wird, wenn man versucht, zwischen Geschwindigkeit und Qualität zu wählen:

Unterer rechter Quadrant: Geschwindigkeit gegenüber Qualität priorisieren
: Das ist die Philosophie »move fast and break things«. Teams, die auf Geschwindigkeit optimieren und dafür die Qualität opfern, schaffen chaotische, fragile Systeme. Sie rutschen in den linken unteren Quadranten, weil sie von ihren schlampigen Systemen ausgebremst werden. Viele Start-ups, die eine Zeitlang so gearbeitet haben, beschweren sich darüber, ihr »Mojo« zu verlieren. Einfache Änderungen, die sie in der guten alten Zeit mal eben rausgehauen hätten, brauchen jetzt Tage oder Wochen, weil das System ein verworrenes Chaos ist.

Oberer linker Quadrant: Qualität gegenüber Geschwindigkeit priorisieren
: Auch bekannt als »wir haben es hier mit ernsthaften und wichtigen Dingen zu tun, daher müssen wir es *richtig* machen«. Der Druck von Deadlines führt

1 Ja, ich arbeite als Berater, warum fragen Sie?

dann zu »Workarounds«. Aufwendige Prozesse schaffen Hürden für Verbesserungen, daher wachsen die technischen Schulden zusammen mit der Liste der »bekannten Probleme«. Diese Teams rutschen in den linken unteren Quadranten. Sie landen bei Systemen schlechter Qualität, *weil* es zu schwer ist, sie zu verbessern. Als Reaktion auf Fehler führen sie noch mehr Prozesse ein. Diese Prozesse machen es noch schwerer, Verbesserungen umzusetzen, und sorgen so für zusätzliche Fragilität und Risiken. Das führt zu mehr Fehlern und mehr Prozessen. Viele Personen, die in so agierenden Organisationen tätig sind, gehen davon aus, dass das normal ist[1] – insbesondere, wenn sie in risikosensiblen Branchen unterwegs sind.[2]

Der obere rechte Quadrant ist das Ziel moderner Ansätze wie Lean, Agile und DevOps. Sich schnell bewegen und ein hohes Qualitätsniveau beibehalten zu können, scheint wie ein Traum zu wirken. Aber die *Accelerate*-Forschung zeigt, dass viele Teams das erreichen. Daher finden Sie in diesem Quadranten »High Performer«.

Die Four Key Metrics

DORAs *Accelerate*-Forschungsteam hat vier zentrale Metriken für die Performance der Softwareauslieferung und von Operations identifiziert.[3] Dafür wurden diverse Messwerte begutachtet, und es wurde herausgefunden, dass diese vier die stärkste Korrelation dazu haben, wie gut eine Organisation ihre Ziele erreicht:

Auslieferungsdurchlaufzeit
: Die Zeit, die erforderlich ist, um Änderungen am Produktivsystem zu implementieren, zu testen und auszuliefern.

Deployment-Häufigkeit
: Wie oft Sie Änderungen an Produktivsystemen deployen.

Anteil an Änderungsfehlschlägen
: Welcher Prozentsatz an Änderungen entweder einen Service beeinträchtigt hat oder eine sofortige Korrektur erforderte, wie zum Beispiel ein Rollback oder ein Notfall-Fix.

Mean Time to Recovery (MTTR)
: Wie lange es dauert, einen Service wiederherzustellen, wenn es einen ungeplanten Ausfall oder eine Beeinträchtigung gibt.

---

1 Das ist ein Beispiel für die »Normalisierung der Abweichung« – die Menschen gewöhnen sich an eine Arbeitsweise, die das Risiko ansteigen lässt. Diane Vaughan hat diesen Begriff in *The Challenger Launch Decision* (University of Chicago Press) definiert.

2 Es ist ironisch (und beängstigend), dass so viele Mitarbeiterinnen und Mitarbeiter in Branchen wie dem Finanzwesen, in Regierungen und in der Gesundheitsversorgung fragile IT-Systeme – und Prozesse, die deren Verbesserung behindern – als normal, ja sogar als wünschenswert ansehen.

3 DORA, jetzt Teil von Google, ist das Team hinter dem *Accelerate State of DevOps Report* (*https://oreil.ly/ysk9n*).

Organisationen, die ihre Ziele gut erreichen – seien es der Umsatz, der Aktienpreis oder andere Kriterien – funktionieren auch gut in Bezug auf diese vier Metriken und umgekehrt. Die Ideen in diesem Buch sollen Ihrem Team und Ihrer Organisation dabei helfen, diese Metriken gut zu erfüllen. Drei zentrale Praktiken für Infrastructure as Code können Sie dabei unterstützen.

Drei zentrale Praktiken für Infrastructure as Code

Das Konzept des Cloud-Zeitalters nutzt die dynamische Natur moderner Infrastruktur- und Anwendungsplattformen aus, um häufig und zuverlässig Änderungen durchführen zu können. Infrastructure as Code ist ein Ansatz, mit dem Sie Infrastruktur bauen, die kontinuierliche Änderungen unterstützt, um eine hohe Zuverlässigkeit und Qualität zu erreichen. Wie kann Ihr Team das schaffen?

Es gibt drei zentrale Praktiken beim Implementieren von Infrastructure as Code:

- Definieren Sie alles als Code.
- Testen Sie all Ihre aktuelle Arbeit kontinuierlich und liefern Sie sie aus.
- Bauen Sie kleine, einfache Einheiten, die Sie unabhängig voneinander austauschen können.

Ich werde jede dieser Thesen nun vorstellen, um den Rahmen für weitere Diskussionen vorzugeben. Im weiteren Verlauf des Buchs werde ich dann jeweils in einem eigenen Kapitel die Prinzipien für das Implementieren dieser drei Praktiken ausführlich erläutern.

Zentrale Praktik: Definieren Sie alles als Code

Es ist eine zentrale Praktik für zügige und zuverlässige Änderungen, alles »als Code« zu definieren. Es gibt dafür ein paar Gründe:

Wiederverwendbarkeit
: Definieren Sie etwas als Code, können Sie viele Instanzen davon erzeugen. Sie können Ihre Elemente reparieren und schnell neu bauen, und andere Personen können identische Instanzen des Elements erzeugen.

Konsistenz
: Elemente, die aus Code gebaut werden, werden jedes Mal gleich gebaut. Dadurch wird das Verhalten von Systemen vorhersagbar, das Testen zuverlässiger und es werden kontinuierliches Testen und Ausliefern möglich.

Transparenz
: Alle können sehen, wie ein Element gebaut wird, indem sie sich den Code anschauen. Die Leute können den Code begutachten und Verbesserungen vorschlagen. Sie können daraus lernen, um Elemente im eigenen Code einzusetzen, Einblicke für das Troubleshooting liefern und zu Compliance-Zwecken Reviews und Audits durchführen.

Ich werde in Kapitel 4 genauer auf die Konzepte und Implementierungs-Prinzipien zum Definieren von Elementen als Code eingehen.

Zentrale Praktik: Kontinuierliches Testen und die gesamte aktuelle Arbeit ausliefern

Effektive Infrastruktur-Teams sind beim Testen rigoros. Sie nutzen Automation, um jede Komponente ihres Systems zu deployen und zu testen, und sie integrieren die gesamte aktuelle Arbeit. Sie testen schon während der Arbeit, statt zu warten, bis sie fertig sind.

Die Idee ist, *Qualität einzubauen*, statt zu versuchen, die *Qualität zu testen*.

Dabei wird gerne übersehen, dass dazu das Integrieren und Testen *der gesamten aktuellen Arbeit* gehört. Bei vielen Teams arbeiten die Entwicklerinnen und Entwickler am Code in eigenen Branches und integrieren erst, wenn sie fertig sind. Laut den Forschungsergebnissen von *Accelerate* liefern Teams allerdings bessere Ergebnisse, wenn sie ihre Arbeit mindestens täglich integrieren. Zum CI gehört das Mergen und Testen des Codes aller Beteiligten während des Entwickelns. CD geht noch einen Schritt weiter und sorgt dafür, dass der gemergte Code immer bereit für die Produktivumgebung ist.

Details zum kontinuierlichen Testen und Ausliefern von Infrastruktur-Code finden Sie in Kapitel 8.

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

Teams bekommen Probleme, wenn ihre Systeme groß und eng gekoppelt sind. Je größer ein System ist, desto schwieriger wird es, es zu ändern, und umso einfacher kann man es zerstören.

Schauen Sie sich die Codebasis eines leistungsfähigen Teams an, sehen Sie den Unterschied. Das System ist aus kleinen, einfachen Elementen aufgebaut. Jedes Element lässt sich leicht verstehen, und es besitzt sauber definierte Schnittstellen. Das Team kann jede Komponente für sich einfach ändern und jede Komponente isoliert deployen und testen.

Genauer gehe ich auf die Implementierungsprinzipien dieser zentralen Praktik in Kapitel 15 ein.

Zusammenfassung

Um etwas aus der Cloud- und Infrastruktur-Automation herauszuholen, benötigen Sie eine Mentalität, die zum Cloud-Zeitalter passt. Sie müssen dafür die Geschwindigkeit ausnutzen, um die Qualität zu verbessern, und Qualität einbauen, um Geschwindigkeit aufzunehmen. Es ist Arbeit, Ihre Infrastruktur zu automatisieren, insbesondere wenn Sie erst lernen, wie Sie das tun können. Aber es hilft Ihnen dabei, Änderungen vorzunehmen – und vor allem, das System überhaupt zu bauen.

Ich habe die Elemente eines typischen Infrastruktur-Systems beschrieben, da diese die Grundlage für die Kapitel legen, in denen erklärt wird, wie Sie Infrastructure as Code implementieren.

Schließlich habe ich drei zentrale Praktiken für Infrastructure as Code beschrieben: Definieren Sie alles als Code, testen und liefern Sie kontinuierlich und bauen Sie kleine Elemente.