1 Einleitung

1.1 Überblick über Continuous Delivery und das Buch

Continuous Delivery ermöglicht es, Software schneller und mit wesentlich höherer Zuverlässigkeit in Produktion zu bringen als bisher. Grundlage dafür ist eine Continuous-Delivery-Pipeline, die das Ausrollen der Software weitgehend automatisiert und so einen reproduzierbaren, risikoarmen Prozess für die Bereitstellung neuer Releases darstellt.

Woher kommt der Begriff Continuous Delivery?

Das agile Manifest *(http://agilemanifesto.org)* definiert als wichtigstes Ziel:

»Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.«

Die deutsche Übersetzung findet sich ebenfalls auf der Website:

»Unsere höchste Priorität ist es, den Kunden durch frühe und kontinuierliche Auslieferung wertvoller Software zufrieden zu stellen.«

Also ist Continuous Delivery eine Technik aus dem agilen Umfeld.

Dieses Buch erläutert, wie eine solche Pipeline praktisch aufgebaut werden kann und welche Technologien dazu eingesetzt werden können. Dabei geht es nicht nur um das Kompilieren und die Installation der Software, sondern auch um verschiedene Tests, die dazu dienen, die Qualität der Software abzusichern.

Das Buch zeigt außerdem, welche Auswirkungen Continuous Delivery auf das Zusammenspiel zwischen Entwicklung (Development) und Betrieb (Operations) im Rahmen von DevOps hat. Schließlich werden die Auswirkungen auf die Softwarearchitektur beschrieben. Neben der Theorie wird ein möglicher Technologie-Stack vorgestellt, der Build, Continuous Integration, Lasttests, Akzeptanztests und Monitoring abdeckt. Für die einzelnen Bestandteile des Technologie-Stacks gibt es jeweils ein Beispielprojekt, mit dem der Leser praktische

Erfahrungen sammeln kann. Das Buch bietet einen Einstieg in den Technologie-Stack und zeigt außerdem auf, wie man sich mit den Themen tiefergehend beschäftigen kann.

Durch Experimente und Vorschläge zum selber Ausprobieren lädt es zur weiteren praktischen Vertiefung ein. Die Leser erhalten so Anregungen, wie sie sich weiter in die Themen vertiefen und eigene Erfahrungen sammeln können. So können die Beispielprojekte Basis für eigene Experimente oder für den Aufbau einer eigenen Continuous-Deployment-Pipeline sein.

Unter *http://continuous-delivery-buch.de* steht die Website zum Buch bereit mit Informationen, Errata und Links zu dem Beispiel.

1.2 Warum überhaupt Continuous Delivery?

Warum sollte man überhaupt Continuous Delivery einsetzen? Eine kleine Geschichte soll diese Frage beantworten – ob sie wahr ist oder nicht, bleibt offen.

Eine kleine Geschichte

Das Marketing eines Unternehmens – nennen wir es Raffzahn Online Commerce GmbH – hatte entschieden, den Registrierungsprozess auf der E-Commerce-Website zu überarbeiten. Dadurch sollten mehr Kunden gewonnen und der Umsatz erhöht werden. Also machte sich ein Entwicklerteam an die Arbeit. Nach nicht allzu langer Zeit waren sie fertig.

Zunächst mussten die Änderungen getestet werden. Dazu hatte das Team der Raffzahn Online Commerce GmbH in einem aufwendigen Prozess eine Testumgebung aufgebaut, auf der die Software manuell getestet wurde. Und es wurden tatsächlich Fehler gefunden. Mittlerweile waren die Entwickler aber schon bei dem nächsten Projekt und mussten sich wieder einarbeiten, bevor sie die Fehler mit einem Fix beheben konnten. Und wegen der manuellen Tests stellte sich bei einigen »Fehlern« heraus, dass die Tester nicht richtig getestet hatten oder die Fehler aus irgendwelchen Gründen nicht reproduzierbar waren.

Nun galt es, den Code in Produktion zu bringen. Der Prozess dazu war aufwendig – denn die E-Commerce-Website der Raffzahn Online Commerce GmbH war über die Jahre gewachsen und daher sehr komplex. Nur dieses eine Feature auszuliefern, rechtfertigte den Aufwand nicht. Daher wurde nur einmal pro Monat deployt. Schließlich konnte die Änderung zusammen mit den anderen Änderungen aus dem letzten Monat in Produktion gehen. Dazu war eine Nacht reserviert. Leider gab es beim Rollout einen Fehler. Das Team machte sich an die Arbeit, das Problem zu analysieren. Aber das war so schwierig, dass das System am nächsten Morgen nicht zur Verfügung stand. Zu diesem Zeit-

punkt waren die Mitarbeiter übernächtigt und standen unter großem Stress – jede Minute Ausfall kostete bares Geld. Und zurück zur alten Version ging es nicht, weil einige Änderungen im Deployment nicht ohne Weiteres rückgängig gemacht werden konnten. Erst im Laufe des Tages, nach einer ausführlichen Fehleranalyse, konnte eine Task Force das Problem beheben und die Website stand wieder zur Verfügung. Der Fehler war eine Konfigurationsänderung gewesen, die in der Testumgebung vorgenommen, aber bei der Produktionseinführung vergessen worden war.

Also schien alles in Ordnung zu sein – aber es gab einen weiteren Fehler, der zunächst nicht entdeckt wurde. Dieser Fehler hätte eigentlich durch die manuellen Tests gefunden werden sollen. Der Test, der den Fehler gefunden hätte, wurde auch erfolgreich durchgeführt. Aber in der Testphase wurden auch einige Fehler gefixt und dieser Test wurde nur vor den Fixes durchgeführt. Der Fehler wurde erst durch einen der Fixes eingeführt. Nach den Fixes wurde der Test nicht noch einmal durchgeführt – daher konnte der Fehler es bis in die Produktion schaffen.

Am nächsten Tag stellte sich also mehr zufällig heraus, dass die Registrierung für die Website der Raffzahn Online Commerce GmbH gar nicht mehr funktionierte. Das war niemandem aufgefallen und erst, nachdem der erste potenzielle Kunde sich bei der Hotline meldete, wurde das Problem erkannt. Wie viele Registrierungen dieser Ausfall gekostet hatte, konnte leider niemand sagen – dazu fehlten Informationen über die Nutzung der Website. Wie schnell die optimierte Registrierung diesen Nachteil ausgleichen konnte, ist fraglich. So konnte es gut sein, dass die Änderung nicht wie ursprünglich geplant zu mehr Registrierungen, sondern zu weniger geführt hatte. Und außerdem war das neue Release wesentlich langsamer – ein Umstand, mit dem vorher auch niemand gerechnet hatte.

Und so begann die Raffzahn Online Commerce GmbH, die nächsten Features zu implementieren, um in einem Monat wiederum ein Update der Website auszurollen. Was wohl dieses Mal passieren würde?

Continuous Delivery hilft.

Continuous Delivery löst solche Probleme durch verschiedene Maßnahmen:

- Es wird öfter deployt – bis hin zu mehreren Malen pro Tag. Dadurch wird die Zeit, bis ein neues Feature genutzt werden kann, verringert.
- Durch häufige Deployments ist auch das Feedback auf neue Features und Code-Änderungen schneller. Die Entwickler müssen sich

nicht erst wieder darauf besinnen, was sie vor einem Monat implementiert haben.

- Um schneller zu deployen, müssen der Aufbau von Umgebungen und die Tests weitgehend automatisiert werden, da der Aufwand sonst zu hoch ist.
- Die Automatisierung führt zu Reproduzierbarkeit: Wenn die Testumgebung erfolgreich aufgebaut werden kann, dann lässt sich mit demselben Automatismus auch die Produktion aufbauen – und zwar mit derselben Konfiguration. Das Problem durch die Fehlkonfiguration der Produktionsumgebung wäre also nicht aufgetreten.
- Außerdem führt die Automatisierung zu mehr Flexibilität. Testumgebungen können On-Demand aufgesetzt werden. So kann es z.B. bei einem Redesign der Oberflächen zeitlich begrenzt eine separate Testumgebung für Marketing geben. Oder für großangelegte Lasttests können zusätzliche Umgebungen aufgesetzt werden, um eine produktionsnahe Umgebung zu haben, die nach den Tests wieder abgerissen werden, so dass keine dauerhaften Investitionen in Hardware notwendig sind – wenn beispielsweise eine Cloud genutzt wird.
- Automatisierte Tests führen dazu, dass Fehler leichter reproduziert werden können. Da die exakt gleichen Schritte bei jedem Test ausgeführt werden, gibt es auch keine Fehler bei der Testdurchführung.
- Wenn Tests automatisiert sind, können sie öfter ausgeführt werden. Also wäre der Fix durch den gesamten Testprozess gegangen und dieser Fehler nicht erst in Produktion aufgefallen.
- Das Risiko eines neuen Release wird weiter reduziert, indem das Deployment in Produktion so aufgesetzt wird, dass es einen Weg zurück zur alten Version gibt. So wird der Produktionsausfall aus dem Beispiel verhindert.
- Und schließlich sollten die Anwendungen auch ein fachliches Monitoring haben, so dass die Registrierung nicht ausfallen kann, ohne dass es jemand merkt.

Durch Continuous Delivery gewinnt das Business eine schnellere Verfügbarkeit neuer Features und eine zuverlässigere IT. Die Zuverlässigkeit ist auch für die IT nützlich. Nachts oder an Wochenenden unter hohem Stress neue Releases auszurollen und Fehler zu beheben, macht

eben keinen Spaß. Und es ist sicher auch für die IT besser, wenn Fehler durch Tests auffallen und nicht erst in Produktion.

Um Continuous Delivery umzusetzen, gibt es eine Vielzahl an Technologien und Techniken. Continuous Delivery hat Auswirkungen bis hin zur Architektur der Anwendung. Genau um diese Themen geht es in diesem Buch. Am Ende steht ein schnellerer und zuverlässigerer Prozess, um Software in Produktion zu bringen.

1.3 Für wen ist das Buch?

Das Buch wendet sich an Manager, Architekten, Entwickler und Administratoren, die Continuous Delivery als Methodik und/oder DevOps als Organisationsform einführen wollen:

- Manager lernen durch den theoretischen Teil Prozess, Erfordernisse und Vorteile von Continuous Delivery für ihr Unternehmen kennen. Außerdem können sie die technischen Konsequenzen von Continuous Delivery abschätzen.
- Entwickler und Administratoren erhalten eine umfassende Einleitung in die technischen Aspekte und können damit die notwendigen Fähigkeiten erlernen, um Continuous Delivery umzusetzen und eine entsprechende Pipeline aufzubauen.
- Architekten können neben den technischen Aspekten auch die Auswirkung auf die Softwarearchitektur kennenlernen – siehe dazu Kapitel 12.

Das Buch stellt verschiedene Technologien für die Umsetzung von Continuous Delivery vor. Als Beispiel dient ein Java-Projekt. Für einige technologische Bereiche – zum Beispiel für die Umsetzung von Akzeptanztests – sind für andere Programmiersprachen andere Technologien etabliert. Das Buch zeigt an den jeweiligen Stellen Alternativen auf, fokussiert aber auf Java. Technologien zur automatisierten Bereitstellung von Infrastrukturen sind unabhängig von der genutzten Programmiersprache. Das Buch ist besonders gut für Leser geeignet, die im Java-Umfeld aktiv sind – für andere Technologien müssen die Ansätze teilweise vom Leser selber transferiert werden.

1.4 Neu in der 2. Auflage

Die Neuauflage wurde in Bezug auf Werkzeuge wie Docker, Jenkins, Graphite und den ELK-Stack aktualisiert. An neuen Themen sind Docker Compose, Docker Machine, Immutable Server, Microservices

und die Einführung von Continuous Delivery ohne DevOps hinzugekommen. Im Einzelnen:

- Der neue Abschnitt 3.6 beschreibt das Konzept »Immutable Server«, bei dem Server niemals mit Updates versehen werden, sondern unveränderlich sind. Das macht Installationen einfacher reproduzierbar.
- Docker ist ein sehr interessantes Werkzeug für das Deployment von Software. Docker Machine vereinfacht die Installation von Docker-Servern. Der neue Abschnitt 3.5.5 beschreibt Docker Machine. Mit Docker Compose können mehrere Docker Container zusammen installiert werden – das beschreibt der neue Abschnitt 3.5.7.
- Continuous Delivery und DevOps haben viele Synergien – aber Continuous Delivery ist auch ohne DevOps möglich. Das beschreibt der neue Abschnitt 11.4.
- Der neue Abschnitt 12.6 erläutert Microservices-Architekturen und die Beziehung zu Continuous Delivery.
- Der ELK-Stack zum Speichern und zur Analyse von Logs in Abschnitt 9.3 nutzt die aktuelle Version von Elasticsearch (2.1), Logstash (2.1) und Kibana (4.3). Dabei hat sich die Installation, aber auch die Oberfläche geändert – deswegen waren auch Änderungen im Abschnitt »Experimente und selber ausprobieren« notwendig. Die Installation ist nun nicht nur mit Vagrant, sondern auch Docker Machine möglich.
- Das Monitoring mit Graphite in Abschnitt 9.8 ist jetzt wesentlich besser modularisiert: Ein Docker-Container nimmt die Metriken entgegen, ein anderer bietet die Webschnittstelle zur Analyse der Daten. Auch hier kann das Beispiel nun mit Docker Machine statt Vagrant genutzt werden.

In der 2. Auflage wurden die Beispiele auf aktuelle Versionen der Werkzeuge umgestellt. Konkret:

- Die Beispiel-VMs benutzen jetzt Ubuntu 15.04.
- Die Beispielanwendung verwendet Spring Boot 1.3.0.
- Das Chef-Beispiel in Abschnitt 3.3 wurde aktualisiert auf Chef 12, Java 1.8 und Tomcat 7
- Docker aus Abschnitt 3.5 basiert jetzt auf Docker 1.10.
- Selenium für GUI-Tests in Abschnitt 5.4 wird in der Version 2.48.2 verwendet.

- JBehave für textuelle Akzeptanztests in Abschnitt 5.7 bezieht sich auf Version 3.9.5.

1.5 Übersicht über die Kapitel

Das Buch umfasst drei Teile. Der erste Teil legt die Grundlagen für ein Verständnis von Continuous Delivery:

- Kapitel 2 führt den Begriff Continuous Delivery ein und zeigt auf, welche Probleme Continuous Delivery wie löst. Dabei wird eine Einführung in die Continuous-Delivery-Pipeline gegeben.
- Für Continuous Delivery muss Infrastruktur automatisiert bereitgestellt werden – es muss Software auf Servern installiert werden. Kapitel 3 stellt dazu einige Ansätze vor. Chef dient zur Automatisierung von Installationen. Mit Vagrant können Testumgebungen auf Entwicklerrechnern eingerichtet werden. Docker ist nicht nur eine sehr effiziente Virtualisierungslösung, sondern kann auch zur automatisierten Installation von Software dienen. Schließlich wird noch ein Überblick über die Nutzung von PaaS-Cloud-Lösungen (Platform as a Service) für Continuous Delivery gegeben.

Es schließen sich im zweiten Teil Kapitel an, die einzelne Bestandteile einer Continuous-Delivery-Pipeline konkret beschreiben. Neben einer konzeptionellen Erläuterung werden jeweils beispielhaft konkrete Technologien dargestellt, mit denen dieser Teil der Pipeline umgesetzt werden kann:

- Im Mittelpunkt von Kapitel 4 von Bastian Spannberg steht alles, was beim Commit einer neuer Softwareversion geschieht. Build-Werkzeuge wie Gradle oder Maven werden vorgestellt, ein Überblick über Unit-Tests wird gegeben und Continuous Integration mit Jenkins wird beleuchtet. Daran schließen sich statische Code Reviews mit SonarQube und Repositories wie Nexus oder Artifactory an.
- Kapitel 5 zeigt mit JBehave und Selenium einen Ansatz für automatisierte GUI-basierte Akzeptanztests und für textuelle Akzeptanztests.
- Performance deckt das Kapitel 6 durch Kapazitätstests ab. Als Beispieltechnologie wird Gatling genutzt.
- Das explorative Testen (Kap. 7) dient dazu, manuell neue Features und generell Probleme in der Anwendung zu überprüfen.

Diese Kapitel betrachten den Anfang der Continuous-Delivery-Pipeline. Diese Phasen beeinflussen hauptsächlich die Softwareentwicklung. Die weiteren Kapitel stellen vor allem Techniken und Technologien vor, die bei den betriebsnahen Bereichen von Continuous Delivery nützlich sind:

- Kapitel 8 zeigt Vorgehensweisen, die beim Rollout der Software in Produktion nützlich sind, um Risiken zu reduzieren.
- Aus dem Betrieb der Anwendung können zahlreiche Daten erhoben werden, um so Feedback zu bekommen. Kapitel 9 stellt dazu Technologien vor: den ELK-Stack (Elasticsearch – Logstash – Kibana) für Log-Dateien-Analyse und Graphite für Monitoring.

Zu den Technologien aus diesen Kapiteln gibt es jeweils Beispiele zum selber Experimentieren und Nachvollziehen am eigenen Rechner. Dadurch können Leser sehr einfach eigene Erfahrungen sammeln. Dank Infrastrukturautomatisierung sind diese Beispiele auch auf dem eigenen Rechner recht einfach zum Laufen zu bringen.

Schließlich stellt sich die Frage, wie Continuous Delivery eingeführt werden kann und welche Auswirkungen Continuous Delivery hat. Das zeigt der dritte Teil des Buchs:

- Kapitel 10 zeigt, wie Continuous Delivery in einer Organisation eingeführt werden kann.
- DevOps beschreibt das Zusammenwachsen von Betrieb (Ops) und Entwicklung (Dev) zu einer Organisationseinheit (Kap. 11).
- Continuous Delivery hat auch Auswirkungen auf die Architektur der Anwendungen. Diese Herausforderungen diskutiert Kapitel 12.
- Am Ende steht das Fazit in Kapitel 13.

1.6 Pfade durch das Buch

Dieser Abschnitt erläutert die möglichen Lesepfade durch das Buch für die verschiedenen Zielgruppen – also welche Kapitel in welcher Reihenfolge gelesen werden sollten. Die Einführung in Kapitel 2 ist für alle Leser interessant – das Kapitel klärt grundlegende Begriffe und zeigt die Motivation für Continuous Delivery auf.

Die weiteren Kapitel haben unterschiedliche Ausrichtungen:

- Für Techniker, die sich vor allem für die Entwicklung interessieren, sind die Kapitel rund um Commit und Tests interessant. In diesen

Kapiteln geht es neben Entwicklung auch um Qualitätssicherung und Build. Die Kapitel zeigen, wie diese Aufgaben durch Continuous Delivery beeinflusst werden, und geben konkrete Code- und Technologiebeispiele aus dem Java-Bereich.

- Administratoren und Betriebsmitarbeiter sollten sich mit Themen wie dem Deployment, dem Bereitstellen von Infrastrukturen und dem Betrieb auseinandersetzen, die ebenfalls durch Continuous Delivery beeinflusst werden.
- Aus der Managementsicht sind die Einführung von Continuous Delivery und der Zusammenhang zwischen DevOps und Continuous Delivery interessant. Diese beiden Kapitel zeigen die Auswirkungen von Continuous Delivery auf die Organisation.
- Schließlich ist der Architektur und dem Fazit jeweils noch ein Extra-Kapitel gewidmet.

Architekten sind eher breit aufgestellt. Sie werden sicher das Kapitel 12 über Architektur lesen, aber meistens interessieren sich Architekten auch für die technischen Details und die Managementsicht. Daher werden Architekten vermutlich mindestens ausgewählte Kapitel aus diesen Bereichen lesen.

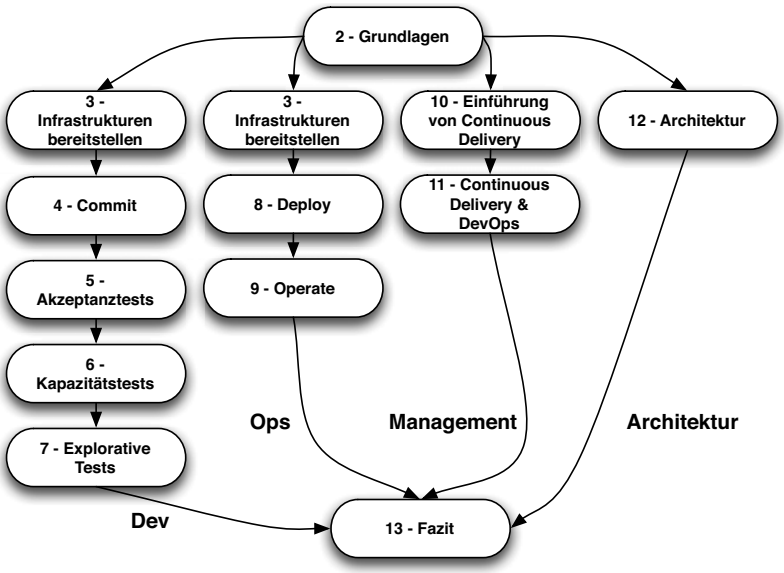

Abb. 1–1
Pfade durch das Buch

1.7 Danksagung

Bedanken möchte ich mich bei Bastian Spannberg für seinen Beitrag zu diesem Buch. Außerdem gilt mein Dank den Reviewern, die mit ihren Kommentaren das Buch wesentlich beeinflusst haben: Marcel Birkner, Lars Gentsch, Halil-Cem Gürsoy, Felix Müller, Sascha Möllering und Alexander Papaspyrou. Bedanken möchte ich mich auch für die vielen Diskussionen – viele Gedanken und Ideen sind in solchen Dialogen entstanden.

Bedanken möchte ich mich auch bei meinem Arbeitgeber, der innoQ.

Schließlich habe ich meinen Freunden, Eltern und Verwandten zu danken, die ich für das Buch oft vernachlässigt habe – insbesondere meiner Frau.

Und natürlich gilt mein Dank all jenen, die an den in diesem Buch erwähnten Technologien gearbeitet haben und so die Grundlagen für Continuous Delivery gelegt haben.

Last but not least möchte ich dem dpunkt.verlag und René Schönfeldt danken, der mich sehr professionell bei der Erstellung des Buchs unterstützt hat.