Mit Stellenmarkt

iX

MAGAZIN FÜR PROFESSIONELLE INFORMATIONSTECHNIK

2 Februar 2012

Februar 2012

€ 6,40
Österreich € 6,70
Schweiz CHF 10,70
Benelux € 7,40
Italien € 7,40

www.ix.de

iX extra Embedded Systems
Funktionale Sicherheit

Für Android und iOS:

Sichere Apps programmieren

Tutorial, Teil II:
C++11
Binder, Callbacks, Lambda-Funktionen

Storage für virtuelle Maschinen:
VMwares SAN-Schnittstelle

Embedded RFID-Chips:
IT-Implantate für den Menschen

Apps für die IT-Abteilung:
Administration mit dem iPad

IT-Sicherheit:
Authentifizierung durch SSL-Handshake

Softwareentwicklung:
Zur Ästhetik der Programmierung

Aus für Tafel und Kreide:
Interaktive Whiteboards

Mit Beiträgen von
innoQ Sonderdruck
Oliver Wolf und Martin Eigenbrodt
innoQ

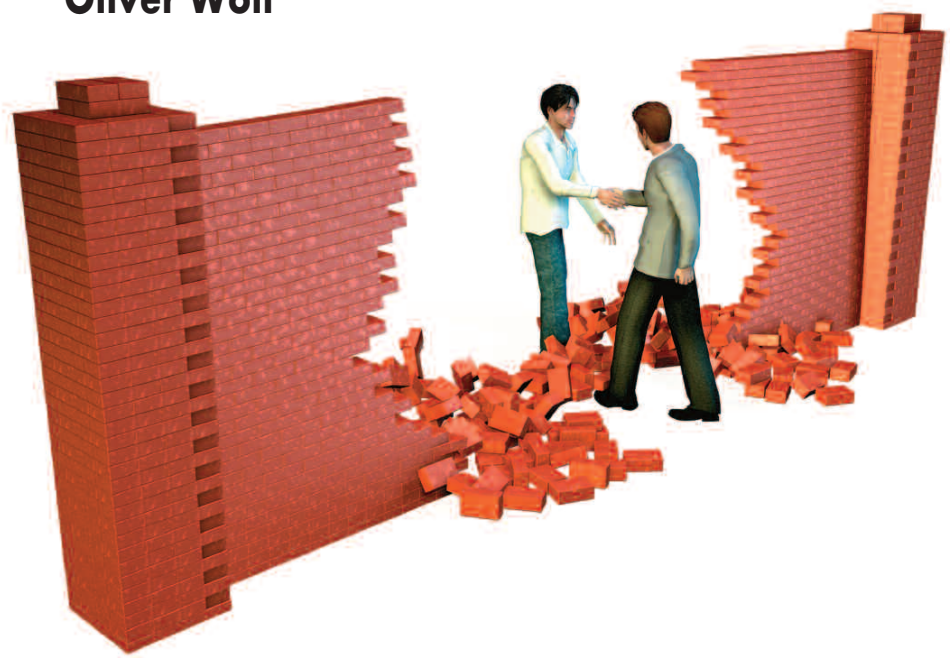
Automatische Bestandserfassung von Hard- und Software:
Inventarisierung mit Open Source

4 199148 506409 02

Entwicklung und Betrieb verzahnen mit DevOps

Wider Betonklötze und Freigeister

Oliver Wolf



DevOps ist auf dem besten Weg, sich zum neuen IT-Hype zu entwickeln. Hinter der noch jungen agilen Bewegung steht die engere Verbindung zwischen Entwicklung und Betrieb. Für die technische Umsetzung haben sich schnell neue Automatisierungswerkzeuge etabliert.

Bis vor einigen Jahren war die IT-Welt zumindest in einem Punkt noch recht übersichtlich: Softwareentwicklung und Systembetrieb waren nahezu immer klar organisatorisch getrennt und oft sogar in unterschiedlichen (Konzern-) Unternehmen angesiedelt. Für diese traditionelle Trennung, noch weiter gefestigt durch die große Outsourcing-Welle in den 1990ern, gab es durchaus gute Argumente. Anders als in der Softwareentwicklung steigt der Personalbedarf im Betrieb nämlich nicht zwangsläufig proportional zur Anzahl der betreuten Systeme, wodurch sich Skaleneffekte beim Konsolidieren des Systembetriebs ergeben. Oder anders ausgedrückt: Es ist für Unternehmen vordergründig wirtschaftlich vorteilhaft, den Betrieb mehrerer Systeme zu bündeln und einer ausgewiesenen Organisationseinheit zu übergeben. Darüber hinaus wird für klassische Betriebs- und Administrationsaufgaben ein anderes Mitarbeiterprofil benötigt als für die Softwareentwicklung, was nahelegt, dass eine organisatorische Trennung keine so schlechte Idee ist.

Naturgemäß ergeben sich durch eine solche Trennung Fragen, die dazu führten, dass der Blick primär auf die Schnittstelle zwischen den beiden Welten gerichtet war. Wie überführt man Software aus der Entwicklung in den Betrieb? Wie lässt sich sicherstellen, dass der Betrieb den Anforderungen der Stakeholder im Unternehmen gerecht wird? Für diese und viele weitere Fragen wurden Antworten gefunden in Form von Prozessen, Metriken, Service Level Agreements (SLAs) und organisatorischen Regelungen, die letztlich in Publikationen wie der IT Infrastructure Library (ITIL) mündeten und bis heute in vielen Organisationen „Gesetz“ sind.

In der Praxis bedeutet das leider allzu oft Reibungsverluste durch Politik – wie es meistens der Fall ist, wenn unterschiedliche Organisationseinheiten mit eigener Agenda nur oberflächlich betrachtet am selben Strang ziehen. Zur Absicherung und zur Verteidigung der eigenen Position legen Betriebsverantwortliche in der Regel viel Wert auf formale Test- und Abnahmeverfahren, oft über viele Stufen hinweg, bis eine Software endlich in Betrieb gehen kann. Aber auch die Entwicklungsseite tut meist viel, um im Zweifelsfall nicht „schuld“ zu sein, wenn im Betrieb etwas schiefläuft. Ausgiebige Tests auf entwicklungsinternen Test- und QS-Plattformen sollen den Nachweis erbringen, dass eigentlich alles funktioniert und Probleme folglich beim Betrieb zu suchen sind.

Es ist offensichtlich, dass diese Art der Zusammenarbeit, auch wenn sie in den meisten Organisationen leidlich funktioniert, sicherlich nicht der Gipfel an Effizienz und Effektivität ist. Abnahmeprozesse und umfangreiche Tests kosten viel Zeit und Geld, vor allem wenn sie auch vertraglich „wasserdicht“ sein sollen. Dementsprechend sind die Releasezyklen bei Inhouse-Entwicklungen oft lang (meistens mindestens 6 bis 12 Monate), damit der für die Betriebsübergaben jeweils notwendige Overhead möglichst selten anfällt. Das wiederum führt aber fast zwangsläufig zu umfangreichen Releases mit einer Vielzahl neuer Funktionen, die somit ein hohes Risiko von Fehlern mit sich bringen, besonders umfangreiche Tests erfordern und vom Betrieb dementsprechend kritisch betrachtet werden. In der klassi-

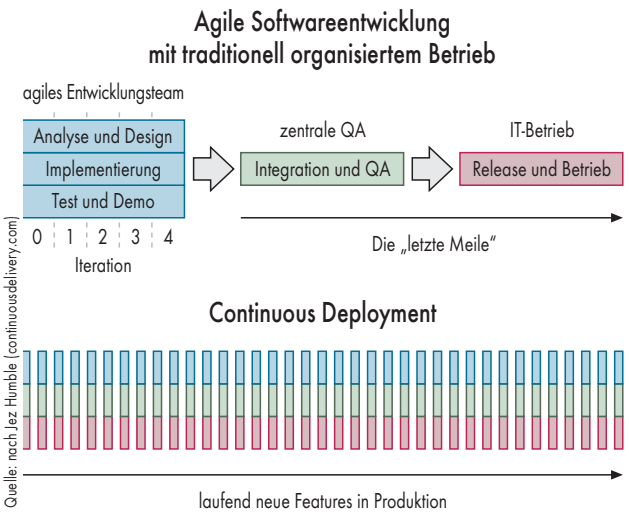
schen Unternehmens-IT hat man sich mit diesem Teufelskreis weitgehend arrangiert – auch wenn den meisten Beteiligten wohl insgeheim bewusst ist, dass hierdurch viel Energie wirkungslos verpufft.

Umdenken durch iterative Verfahren

Spätestens mit der Jahrtausendwende aber ist in vielen Unternehmen, zumindest in der Softwareentwicklung, einiges in Bewegung geraten. War bis dahin das klassische Wasserfallmodell meist noch das Maß der Dinge, hat sich, nicht zuletzt durch eine zunehmende Anzahl gescheiterter IT-Projekte, zunehmend die Erkenntnis durchgesetzt, dass zwischen Plan und Illusion oft nur ein schmaler Grat liegt. Anforderungen an Software lassen sich eben nur mit immensem Aufwand – wenn überhaupt – vollständig erfassen und sind darüber hinaus aufgrund vieler geschäftlicher, gesetzlicher und technischer Faktoren Änderungen unterworfen, die oft erst während der Entwicklung erkennbar werden.

Das führte zu einer weitreichenden Verbreitung iterativer Vorgehensweisen, die in agilen Methoden wie Kanban und Scrum ihren vorläufigen Höhepunkt finden. Es dürfte heute kaum ein Unternehmen mehr geben, das nicht zumindest agile Elemente in seinen Softwareentwicklungsprozess integriert hat. Dabei ist das Ziel klar: Das Risiko, Software an den Anforderungen der Projektauftraggeber vorbei zu entwickeln, soll reduziert und damit letztlich Entwicklungskosten gespart werden. Voraussetzung für eine erfolgreiche Anwendung agiler Methoden ist aber eine enge Einbindung derer, von denen primär die Anforderungen kommen: der Anwender in den Fachabteilungen, deren tägliche Arbeit durch das neue System erleichtert und effizienter gestaltet werden soll.

Wie aber schafft man es, die Anwender mit einzubeziehen, wenn sich aufgrund der beschriebenen Probleme nur ein bis zwei Releases pro Jahr in den produktiven Betrieb nehmen lassen? Eine Möglichkeit besteht darin, Anwender mit einem regelmäßig auf neue Versionen aktualisierten Testsystem „spielen“ und neue Elemente ausprobieren zu lassen. In der Praxis funktioniert das selten gut, weil sich viele Fehler und Schwächen einer komplexen Software eben nur an realen Geschäftsvorfällen mit wirklichen Daten offenbaren und darüber hinaus der Zeitaufwand für die Anwender, neben ihrer täglichen Arbeit auch noch neue Funktionen zu testen, oft inakzeptabel hoch ist. Eine andere, in der Praxis meist bevorzugte Option ist es, eine neue Rolle zu schaffen, die sich – je nach Organisation – Produktmanager, User-Proxy oder ähnlich nennt und die verantwortungsvolle Aufgabe hat, sich in die Lage der Endanwender zu versetzen und für sie über den Einsatz der Funktionen zu entscheiden. Den entsprechenden fachlichen Hintergrund vorausgesetzt, kann das bei einer überschaubaren Anzahl von Anwendern mit relativ homogenem Nutzungsprofil (z. B. 200 Schadenssachbearbeiter einer Versicherung, die nach einem definierten Prozess arbeiten) durchaus gut funktionieren.



Mit häufigen kleinen Releases lassen sich Risiken und der Nutzen durch Feedback ausbalancieren.

Web 2.0 als Antreiber

Anders sieht die Situation bei einer neuen Klasse von Anwendungen aus, die seit Mitte des letzten Jahrzehnts unter dem Stichwort „Web 2.0“ den Markt erobert. Egal ob Auktionsplattform, Foto- und Video-Sharing-Dienst oder Social-Media-Plattform: Diesen Anwendungen ist gemeinsam, dass ihre Zielgruppe um mehrere Größenordnungen größer ist als die von Unternehmensanwendungen – und darüber hinaus global verteilt und in jeder Hinsicht extrem inhomogen. Kein Produktmanager hat hierbei eine realistische Chance, ohne ständiges und direktes Anwenderfeedback ein nachhaltig attraktives Angebot zu gestalten. Und gerade Nachhaltigkeit ist ein entscheidender Punkt, denn anders als bei unternehmensinternen Anwendungen haben die Nutzer die freie Wahl und wandern schnell zum Wettbewerbsprodukt ab, wenn sie ihre Anforderungen nicht zeitnah erfüllt sehen.

Generell ist Time-to-Market für solche Anwendungen der wesentliche Erfolgsfaktor, denn im Gegensatz zum ersten Internet-Boom Ende der 1990er sind Investoren nach dem Platzen der Dotcom-Blase im Frühjahr 2000 deutlich vorsichtiger geworden, und die Bereitschaft, in pure Visionen zu investieren, ist spürbar gesunken. Ohne zumindest ein minimal lauffähiges Produkt vorweisen zu können, ist Kapital, wenn überhaupt, heute meist nur zu unattraktiven Konditionen zu bekommen, sodass Start-up-Gründer erhebliche Vorleistungen erbringen müssen. Ziel ist es in der Regel, so schnell wie möglich mit einem von der Grundidee her innovativen und überzeugenden Produkt auf den Markt zu gehen und dieses dann, wiederum basierend auf dem Anwenderfeedback und den Anforderungen

X-TRACT

- DevOps steht für eine engere Verbindung zwischen Entwicklungsabteilung und Systembetriebsseite unter Zuhilfenahme agiler Verfahren und automatisierter Praktiken.
- Die Voraussetzung für DevOps ist eine entsprechende Kultur im Umgang miteinander sowie der Einsatz von Werkzeugen, mit denen sich Betriebsaufgaben automatisieren lassen.
- Traditionell aufgestellte Unternehmen betrachten DevOps derzeit oft noch skeptisch und nutzen die Vorteile damit einhergehender Werkzeuge nur zögerlich.

durch steigende Benutzerzahlen, zügig zu erweitern und zu verbessern.

Mit dieser neuen Klasse von Anwendungen hat eine neue Offenheit in der Kommunikation Einzug gehalten. Anbieter von Internetanwendungen geben in ihren Technik-Blogs und auf einschlägigen Konferenzen nicht ohne Stolz bereitwillig Einblick in ihre Systemarchitektur und die Agilität ihrer Änderungsprozesse. Beispielsweise berichtet die Blogging-Plattform WordPress von 25 000 Releases in viereinhalb Jahren (ca. 16 Releases pro Tag) [a], die E-Commerce-Plattform Etsy bringt es nach eigenen Angaben gar auf 25 Releases pro Tag [b]. Angesichts solcher Zahlen ist sowohl offensichtlich, dass der Begriff „Release“ hier eine andere Bedeutung haben muss als in der klassischen Unternehmens-IT, als auch, dass die Prozesse für die Produktivsetzung von Änderungen fundamental anders sein müssen.

„What Is This Devops Thing?“

In der Tat ist es so, dass praktisch alle diese Unternehmen auf die klassische organisatorische Trennung von Softwareentwicklung und Systembetrieb weitgehend verzichten. Ursprünglich sicherlich auch durch Sachzwänge wie geringes Budget und entsprechend kleine Teamgrößen begründet, hat sich daraus im Laufe der Zeit eine regelrechte Bewegung formiert, die den Schulterschluss zwischen Entwicklung (Development) und Betrieb (Operations) zum Programm gemacht hat: DevOps. Der Begriff wurde erstmals durch den Belgier Patrick Debois [c] und die von ihm 2009 initiierte Fachkonferenz Devopsdays [d] geprägt, die damals weitgehend unbeachtet den Startschuss für eine Vielzahl von Fachartikeln, Konferenzbeiträgen und Blogs gegeben hat.

Auch wenn die DevOps-Bewegung bislang noch viele Gesichter hat und eine eindeutige Definition schwerfällt, ist doch die wesentliche Erkenntnis, die sich aus diesen Quellen gewinnen lässt, Folgende: Die Voraussetzungen dafür, dass eine enge Zusammenarbeit von Entwicklung und Betrieb in der Praxis funktioniert, sind eine entsprechende Kultur im Umgang miteinander sowie die Verfügbarkeit (und der Einsatz) von Werkzeugen, mit denen sich Betriebsaufgaben automatisieren lassen.

Der erste Faktor, die Kultur, ist im Kern fast trivial, in der Praxis aber in vielen Organisationen noch eine große Hürde. Im Prinzip geht es darum, das Verständnis dafür zu schaffen, dass beide Parteien – Entwicklung und Betrieb – gleichermaßen zum Erfolg eines Unternehmens beizutragen haben und das Gelingen eines Projekts letztlich von beiden abhängt. Beispielsweise helfen gegenseitige Schuldzuweisungen im Fehlerfall niemandem weiter und sind, im Gegenteil, kontraproduktiv, weil wertvolle Zeit und Energie für Rechtfertigungen und Absicherungen verloren geht, die für die Lösung des eigentlichen Problems fehlt. Dazu kommt es zu oft stereotypen Vorstellungen darüber, wie die Kollegen „auf der anderen Seite“ denken und handeln, was wiederum zu mangelndem Respekt und Vertrauen führen kann.

Auch heute gibt es durchaus noch Systemadministratoren, für die Softwareentwickler Freigeister ohne Qualitätsanspruch sind, deren Aufgabe mit dem intellektuell anspruchsvollen Teil der Lösung des Problems endet und die sich um betriebliche Aspekte nur am Rande kümmern (wollen). Genauso besteht bei so manchem Entwickler das Bild vom Systemadministrator als risikoscheuem „Betonkopf“, der sich nicht ins Handwerk pfuschen lassen will. Auch wenn an solchen Vorstellungen sicherlich immer ein kleiner Funken Wahrheit ist, sind sie doch glücklicherweise mehrheitlich unzutreffend. Dennoch sind Schuldzuweisungen und mangelndes Vertrauen in vielen Unternehmen an der Tagesordnung, was in aller Regel weitaus weniger ein Problem der handelnden Personen selbst als vielmehr der organisatorischen Ausrichtung des Unternehmens sein dürfte.

Wenn der Systembetrieb an Kriterien wie Stabilität und „Uptime“ gemessen wird, die Softwareentwicklung aber an der Erfüllung funktionaler Anforderungen und umgesetzter Features pro Zeiteinheit, resultiert das zunächst einmal in einem nicht ohne Weiteres überwindbaren Zielkonflikt, der sich durch reine Absichtserklärungen („Habt euch doch lieb!“) nicht auflösen lässt. Fast zwangsläufig muss ein kultureller Wandel einsetzen, der auf der Erkenntnis fußt, dass Agilität nicht bei der Softwareentwicklung enden darf, wenn sie nachhaltigen Nutzen bringen soll.

Werkzeuge und Praktiken

Damit Agilität an der Schnittstelle zum Betrieb und darüber hinaus funktioniert, braucht es die zweite Komponente von dem, was DevOps ausmacht: geeignete Werkzeuge und Praktiken. Denn wenn die Häufigkeit von Änderungen zunimmt – und genau darum geht es ja – nimmt zwangsläufig auch die Häufigkeit von Fehlern zu. Menschen begehen Fehler, das lässt sich nicht verhindern. Was sich aber ändern lässt, ist der Umgang mit Fehlern. In der Softwareentwicklung hat sich inzwischen auf breiter Front die Ansicht durchgesetzt, dass viele kleine Änderungen besser beherrschbar sind und weniger Risiken bergen als wenige große Änderungen. Mit Praktiken wie Continuous Integration (CI) trägt man dieser Erkenntnis dort Rechnung und gibt jedem Entwickler die Möglichkeit, die Auswirkungen seiner Änderung unmittelbar zu sehen und entsprechend zu reagieren („fail fast, fail early, fail often“). Dazu ist es notwendig, das in Entwicklung befindliche System automatisiert, das heißt ohne zusätzliche potenzielle Fehlerquellen durch manuelle

Onlinequellen

[a] Toni Schneider; In praise of continuous deployment: The WordPress.com story toni.org/2010/05/19/in-praise-of-continuous-deployment-the-wordpress-com-story/

[b] Chad Dickerson; How does Etsy manage development and operations? codeascraft.etsy.com/2011/02/04/how-does-etsy-manage-development-and-operations/

[c] Patrick Debois; What Is This Devops Thing, Anyway? jedi.be/blog/2010/02/12/what-is-this-devops-thing-anyway/

[d] Devopsdays devopsdays.org

[e] Chef www.opscode.com/chef/

[f] Puppet puppetlabs.com/

[g] Capistrano github.com/capistrano/capistrano/wiki/

[h] Fabric fabfile.org

[i] RunDeck rundeck.org

[j] Cobbler fedorahosted.org/cobbler/

[k] Vagrant vagrantup.com

Schritte, möglichst weitgehend zu testen.

Durch die Entfernung des Faktors „Mensch“ aus der Prozesskette entsteht Vertrauen in das korrekte Funktionieren der Werkzeuge, und damit wächst die Bereitschaft, notwendige Änderungen durchzuführen. Diese Idee weiterzuführen bis in die Betriebsumgebung hinein ist einerseits naheliegend, andererseits aber geradezu revolutionär. Denn genau an der traditionellen Schnittstelle zwischen Entwicklung und Betrieb verschwimmen nun die Grenzen. Wer entscheidet, ob eine Release gut genug ist für die Produktion? Wem „gehört“ das Build- und Deployment-System? Man sieht schnell, dass traditionelles Revierdenken hier keinen Sinn mehr ergibt. Die Erweiterung von Continuous Integration in den Systembetrieb hinein wird auch als Continuous Delivery oder Continuous Deployment bezeichnet [2] und ist eine der Kernpraktiken von DevOps. Nur durch häufige, kleine Releases – und hier sieht man, dass dieser Begriff anders zu verstehen ist als in der klassischen Unternehmens-IT – lassen sich Risiken durch Fehler und der Nutzen durch zeitnahes Feedback realer Endnutzer ausbalancieren.

Damit das in der Praxis technisch umsetzbar ist, werden neue Automatisierungswerkzeuge benötigt, die über den Umfang herkömmlicher Build-Systeme für die Softwareentwicklung wie Ant und Maven deutlich hinausgehen. Mindestens drei grobe Kategorien solcher Werkzeuge haben sich in der letzten Zeit herausgebildet.

Die erste Kategorie dient der Automatisierung allgemeiner Aufgaben in der Systemadministration. Die Idee dahinter ist nicht neu – fast jeder Systemadministrator hat seinen eigenen Satz von Skripten, um immer wiederkehrende Administrationsaufgaben effizient und möglichst fehlerfrei zu erledigen. Das Problem an der althergebrachten Vorgehensweise ist jedoch die Schwierigkeit der Kooperation zwischen Entwicklung und Betrieb. Nicht jeder Softwareentwickler ist auch ein begnadeter Shell-Programmierer beziehungsweise kennt sich mit den letzten Tiefen von Unix-Systemen aus. Auf der anderen Seite kann ein Systemadministrator nicht wissen, was genau auf dem Zielsystem passieren muss, um dieses für eine bestimmte Version einer Software zu konfigurieren. Zusammenarbeit ist demnach zwingend notwendig und geschieht klassisch über Betriebsdokumente, in denen der Entwickler das Vorgehen beschreibt und die der Administrator in ausführende Skripte umsetzt. Besser und effizienter wäre es, wenn beide Seiten ein gemeinsames Vokabular hätten, um die notwendigen Schritte so zu beschreiben, dass sie direkt als Programm ausführbar sind. Diesen Ansatz bezeichnet man auch als „Infrastructure as Code“, und mit Chef [e] und Puppet [f] existieren zwei leistungsfähige Werkzeuge, die jeweils eine für beide Seiten verständliche domänenspezifische Sprache (DSL) verwenden. Wie Continuous Delivery ist auch „Infrastructure as Code“ eines der Kernkonzepte von DevOps. Im Kasten wird daher darauf noch etwas detaillierter eingegangen.

Ein großer Vorteil solcher Produkte ist – neben der Verbesserung der Kommunikation – auch die Möglichkeit, die Konfigurationsschritte gemeinsam mit der zu installierenden Software in einem Versionskontrollsystem abzulegen. Dadurch ist es leicht zu realisieren, mehrere Systeme aus einer einheitlichen Quelle heraus in kurzer Zeit betriebsfertig einzurichten, was insbesondere bei massivem Einsatz von Virtualisierung beziehungsweise Verwendung von Rechenressourcen in der Cloud nützlich ist. Neben Chef und Puppet gibt es noch weitere Vertreter dieser Kategorie, die aber aufgrund ihrer geringe-

Werkzeuge für „Infrastructure as Code“

Chef: Automatisierung nach Rezept

Grundidee von Chef ist die Organisation miteinander in Zusammenhang stehender Administrationsvorgänge in Form von Chef-Programmen, die als Recipes (Kochrezept) bezeichnet werden. Ein solches Recipe könnte zum Beispiel beschreiben, was zu tun ist, um aus einem „blanken“, nur mit einer Betriebssysteminstallation ausgestatteten Server einen Applikationsserver für eine bestimmte Anwendung zu machen. Ein Recipe wiederum besteht aus einer Menge sogenannter Resources, die plattformunabhängig beschreiben, was genau auf dem Zielsystem zu konfigurieren ist.

Resources sind rein deklarativ, das heißt, sie beschreiben nur, was zu tun ist, nicht den technischen Ablauf dahinter. Das ist Aufgabe der sogenannten Provider, die die für eine Resource notwendigen konkreten Schritte plattformspezifisch kapseln.

Ein Beispiel: Eine konkrete Art von Resource ist die Package Resource, über die sich ein Softwarepaket wie Tomcat installieren lässt. Da es je nach Zielplattform unterschiedliche Wege gibt, diese Installation auszuführen, existieren im Lieferumfang von Chef verschiedene Provider für diese Resource, unter anderem zur Steuerung der Package-Manager rpm, apt und yum. Durch die Abstraktion von der konkreten Zielplattform lassen sich so komplexe Administrationsvorgänge ohne detailliertes Plattformwissen beschreiben und dann in einer (auch heterogenen) Umgebung ausführen.

Neben Recipes und Resources gibt es noch weitere Konzepte in Chef, die insbesondere für komplexere Umgebungen wichtig sind: Attributes erlauben die Parametrierung von Resources für bestimmte Zielsysteme. Damit lässt sich etwa die HTTP-Port-Adresse eines installierten Applikationsservers variieren. Über Roles kann man Recipes zusammenfassen und als Einheit auf ein Zielsystem anwenden. Zum Beispiel könnte ein System mit der Role „MyApp“ die Recipes für die Installation des Applikationsservers und das Deployment der Applikation „MyApp“ umfassen. Die DSL von Chef basiert auf der Programmiersprache Ruby und erlaubt dadurch die Vermischung mit freiem Ruby-Code, wodurch Chef zu einem mächtigen Werkzeug wird.

Je nach gewünschtem Einsatzzweck gibt es das Werkzeug in unterschiedlichen Ausprägungen. Die einfache Version Chef solo läuft auf einem Rechner und führt Chef-Programme lokal aus. Damit eignet sie sich gut für einen Einstieg ins Thema und die Entwicklung von Chef-Programmen. In realen Systemen wird man eher Chef Client/Server einsetzen, bei dem der Anwender lokale Agenten über einen zentralen Server gesteuert aktiviert und das jeweilige System entsprechend den zentral abgelegten Vorgaben (um-)konfiguriert.

Puppet: Der Administrator als Drahtzieher

Puppet ist das etwas ältere (und damit schon etabliertere) der beiden Systeme. Es folgt einem ähnlichen Ansatz wie Chef. Auch Puppet ist in der Programmiersprache Ruby entwickelt, seine primäre, von den meisten Anwendern eingesetzte DSL ist aber eine sogenannte externe DSL. Das heißt, Puppet-Programme sind keine Ruby-Programme. Dadurch ist Puppet in den Ausdrucksmöglichkeiten etwas eingeschränkter, was aber in der Praxis manchmal eher Vorteile durch eine klarere Struktur der Programme hat. Ansonsten ist der Unterschied zu Chef fast schon philosophischer Natur: Puppet-Programme beschreiben, wie ein System zu *sein* hat, während Chef-Recipes festlegen, was zu *tun* ist. Ein kleiner, in der Praxis aber manchmal durchaus relevanter Vorteil der Puppet-Philosophie ist die bessere Handhabung von Abhängigkeiten zwischen Konfigurationsobjekten.

Die Qual der Wahl

Sowohl Chef als auch Puppet erfreuen sich großer Beliebtheit und können aktive Anwender-Communitys und eine umfangreiche Dokumentation vorweisen. Vorgefertigte Recipes (bzw. Module bei Puppet) für unterschiedliche Zwecke sind zahlreich vorhanden, sodass der Aufwand für die Automatisierung typischer Administrationsaufgaben in der Regel nicht mehr hoch ist. Traditionell hatte Puppet seine Anhänger eher unter den Systemadministratoren, während Chef vorrangig Entwickler anziehen dürfte, aber letztlich wird jeder nach seinem persönlichen Geschmack entscheiden müssen, welches der Werkzeuge er für seinen Einsatzzweck wählt.

ren Verbreitung wohl eher eine Nebenrolle spielen.

Die zweite Kategorie Werkzeuge ist von der ersten nicht immer scharf abgrenzbar und beschäftigt sich mit dem Deployment von Anwendungen auf Zielsysteme. Wichtige Vertreter sind etwa Capistrano [g], Fabric [h] und RunDeck [i]. Die meisten dieser Tools bewegen sich auf einer niedrigeren Abstraktionsebene als die erste Kategorie und sind – vereinfacht ausgedrückt – eher so etwas wie automatisierte Remote-Shells über SSH.

Eine dritte Kategorie setzt noch früher an und schafft zuerst einmal die Voraussetzungen dafür, dass Werkzeuge der anderen Kategorien überhaupt funktionieren können. Mit Tools wie Cobbler [h] und Vagrant [i] lassen sich (virtuelle) Maschinen automatisiert provisionieren, das heißt technisch bereitstellen und mit einer grundlegenden Betriebssysteminstallation ausstatten.

Fazit

So umfangreich der Werkzeugkasten bei DevOps mittlerweile ist – gerade traditionell aufgestellte Unternehmen betrachten das Thema zurzeit oft noch skeptisch und nutzen die Vorteile solcher Tools nur zögerlich. Das ist auf der einen Seite verständlich, denn ohne eine tiefgreifende Änderung der Organisationskultur, so scheint es auf den ersten Blick, ergeben die Ideen der DevOps wenig Sinn. Auf der anderen Seite kann gerade ein technischer Einstieg über mehr und konsequentere Automatisierung – von welcher Seite auch immer getrieben – neben dem offensichtlichen Effizienzgewinn dabei helfen, Vorbehalte abzubauen sowie Vertrauen in die Prozesse und Methoden hinter den Werkzeugen zu schaffen. In diesem Sinn ist auch in der klassischen Unternehmens-IT genug Platz für DevOps, und es besteht zumindest die Chance, dass irgendwann auch ein positiver Einfluss auf die Kultur zu spüren ist. (ane)

OLIVER WOLF

ist Principal Consultant bei der innoQ Deutschland GmbH. Neben Softwarearchitektur und agilen Softwareentwicklungsmethoden gilt sein Interesse der engeren Verbindung von Entwicklung und Betrieb.

Literatur

[1] Ingmar Krusch, Schlomo Schapiro; Mauern einreißen; Neuartige Zusammenarbeit zwischen Softwareentwicklung und Betrieb; *iX Developer* 1/2012, S. 144

[2] Carsten Bernhard; Agil bis zum Schluss; Pünktliche Releases mit Continuous Delivery; *iX* 11/2011, S. 84

Alle Links: www.ix.de/ix1202114