JAVA Mag inkl. CD

Open Source BPM: Tools und Strömungen ▸86

S&S

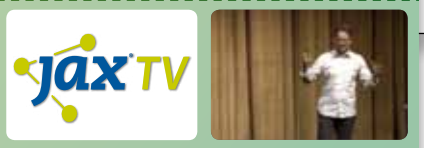
1.2012

Java™ magazin

Java • Architekturen • Web • Agile

www.javamagazin.de

CD-INHALT



DEVOPS-KEYNOTE
von Matthias Marschall

Video von der W-JAX 2011

HIGHLIGHT

Cucumber-chef v1.0.4-0
ChameRIA 1.5.0
jBPM 5.1.0.Final

WEITERE INHALTE

- H-Ubu
- Apache POI 3.8 beta 4
- Activiti 5.8

Alle CD-Infos ab Seite 3

NIO2

Erweiterungen der I/O-Funktionalität ▸27

Product Line Engineering mit OSGi

Ein sinnvolles Paar? ▸105

jax 2012

Erste Sessions ab Seite ▸43

DevOps

Entwicklung und Betrieb zusammenbringen ▸▸ 32, 41

Infrastructure as Code mit Chef ▸▸ 53

Mit Beiträgen von

innoQ Sonderdruck

Oliver Wolf und Martin Eigenbrodt

innoQ

GWT & HTML5

Das große Web-Tutorial ▸S. 60

Google Dart

Die neue Programmiersprache ▸S. 75

Spring ohne XML

Java-basierte Konfiguration ▸S. 96

Server programmieren statt konfigurieren

„Infrastructure as Code“ mit Chef

Im Rahmen der DevOps-Bewegung setzt sich die Erkenntnis durch, dass das Konfigurieren von Servern ähnlich wie das Entwickeln von Software behandelt werden sollte. Testbare, wiederholt ausführbare Programme ersetzen die manuelle Konfiguration. Dieser Artikel zeigt am Beispiel Chef, wie dieses Paradigma umgesetzt wird.

von Martin Eigenbrodt

Ein typischer erster Arbeitstag in einem neuen Projekt: Es handelt sich um ein ganz „gewöhnliches“ Java-EE-Projekt. Ich bekomme Zugangsdaten für eine Versionsverwaltung und ein Wiki-System. Ich habe Glück, der Code lässt sich sofort bauen. Nun würde ich meine nagelneuen Artefakte gerne deployen und in Aktion sehen. Das bedeutet zunächst: Wiki-Seiten wälzen. Dort finde ich Verweise auf den verwendeten Application-Server und Fragmente für die Konfiguration. Ich wühle mich also durch die Downloadseiten des Herstellers, um die korrekte Version zu finden, und editiere dann die Konfigurationsdateien. Für die verwendete Datenbank bekomme ich vom Kollegen ein Image für eine Virtualisierungssoftware. Allerdings, so warnt er mich, müsse ich einige Schemaänderungen noch per Hand nachpflegen. Etwas später habe ich die Migrationsskripte gefunden und führe sie aus. Per Hand deploye ich die neuen Artefakte und es entstehen merkwürdige Fehler. Die Fehlersuche mit dem Kollegen offenbart, dass ich noch einige obligatorische Verzeichnisse anlegen muss und die Konfigurationsdateien aus dem Wiki nicht ohne Änderungen zu der Datenbank im Image passen. Schlussendlich bin ich in der Lage, das Projekt zum Laufen zu bringen. Aber ob mein System wohl identisch mit dem des Kollegen ist? Oder der Testumgebung? Oder der Produktion? Und mit wie viel Aufwand für den Aufbau einer weiteren Staging-Umgebung muss man rechnen?

Infrastructure as Code

Kommt Ihnen das geschilderte Szenario bekannt vor? Wie aber lässt sich die Situation verbessern? Wir können uns einiges bei uns selbst abschauen: Der Quelltext des Projekts ließ sich ohne Probleme auschecken und erfolgreich bauen. Das hat vor allem zwei Gründe: Erstens verwalten wir Quelltext in einem Versionskontrollsystem (VCS) und Zweitens wird er regelmäßig automatisch gebaut und getestet. Wir können Änderungen vornehmen, ohne befürchten zu müssen, sie nicht mehr rückgängig machen können. Das VCS verrät uns bei richtiger Anwendung, wann, von wem und warum eine Änderung gemacht wurde; und jeder kann jederzeit den aktuellen Stand abrufen. Darüber hinaus ist durch die automatisierten Tests eine gewisse Qualität gesichert. Wenn es uns nun gelänge, alle Schritte zum Aufsetzen der Infrastruktur, hier also der Datenbank, des Applikationsservers und das Deployen der Anwendung mit einer ausführbaren Dokumentation zu versehen, könnte sie ebenfalls im VCS gespeichert und automatisch verifiziert werden.

Dieses Vorgehen wird als „Infrastructure as Code“ bezeichnet. Dabei geht es nicht vorrangig um das Einsetzen eines neuen Tools, sondern um einen Paradigmenwechsel. Das Einrichten von Systemen wird als Programmieren begriffen, und wir übertragen Vorgehensweisen aus der klassischen Entwicklung auf das Entwickeln von Infrastruktur. Lauffähige Systeme werden damit vom statischen Artefakt, das einmal eingerichtet wird, zum Ergebnis eines ausführbaren Programms. Das macht sie duplizierbar und versionierbar.

Einmal geschriebener Code kann in einem virtuellen Rechner auf dem Entwicklercomputer, in der Cloud oder auf echter Hardware ausgeführt werden. Dadurch gelingt es, Unterschiede zwischen Entwickler-, Test- und Produktivsystemen zu verringern. Darüber hinaus wird das lauffähige System zum Wegwerfartikel: Es kann jederzeit gelöscht, neu erstellt oder dupliziert werden. Das erlaubt es zum Beispiel, Systeme nur für die Laufzeit bestimmter Tests zu erstellen, und macht damit einen Hauptvorteil des Cloud Computing, also die kurze Provisionierungsdauer und Abrechnung nach tatsächlich genutzter Zeit, überhaupt erst in größerem Umfang nutzbar. Infrastruktur zu programmieren und versionieren statt nur zu konfigurieren hat Vorteile weit über den geschilderten Anwendungsfall des Einrichtens der Entwicklungsumgebung hinaus.

Chef

Um Infrastruktur zu programmieren, bedarf es einer Programmiersprache, die darauf zugeschnitten ist. Chef stellt eine solche domänenspezifische Sprache (DSL) bereit. Als Framework leistet Chef aber noch mehr: Es sammelt alle Informationen an einer Stelle und bietet ein REST API, um verschiedenste Eigenschaften der Infrastruktur abzufragen. So können Fragen wie „Was ist die IP-Adresse des Webservers für die Umgebung DEV?“ oder „Auf welchem Rechner ist Version X des Programms Y installiert?“ beantwortet werden.

Große und kleine Köche

Chef gibt es in verschiedenen Ausführungen. Als *Chef Solo* läuft es auf einem Computer und führt Chef-Programme aus, die lokal ohne Verbindung vorliegen. Diese einfache Variante eignet sich vor allem für erste Experimente mit Chef. In der Praxis kommt meist *Chef Client/Server* zum Einsatz (**Abb. 1**). Dabei gibt es einen zentralen Server. Auf jedem Rechner, der über Chef konfiguriert werden soll, läuft der Chef-Client in der Regel als Service. Der Chef-Client fragt in konfigurierbaren, regelmäßigen Abständen über das REST API des Servers den gewünschten Zustand ab und konfiguriert das System gegebenenfalls um.

Entwickler oder Systemadministratoren nehmen Veränderungen vor, indem sie mit dem Programm *Knife* Befehle an das REST API des Chef-Servers senden. Neben Knife kann auch eine Weboberfläche verwendet werden, um Chef zu steuern. Langfristig ist man mit Knife aber flexibler und schneller.

Cookbooks

Chef-Programme werden in so genannten Cookbooks (Kochbücher) abgelegt. Ein Cookbook aggregiert verschiedene Objekte, darunter die eigentlichen Programme (so genannte Recipes), Attributdefinitionen und Template-Dateien. All diese Dinge werden innerhalb einer vorgegebenen Verzeichnisstruktur abgelegt, die mit dem Befehl *knife cookbook create name* angelegt werden kann. In der Regel werden die Cookbooks in einer Versionsverwaltung gespeichert (bevorzugt Git, aber auch andere sind möglich) und von dort mit *knife cookbook upload name_des_cookbook* zum Chef-Server übertragen. Eine Liste aller auf dem Chef-Server vorhandenen Cookbooks kann mit *knife cookbook list* abgerufen werden.

Rezepte, Ressourcen und Provider

Recipes sind in der Regel eine Aneinanderreihung von Ressourcen (Resource). Eine Resource ist eine plattformunabhängige Repräsentation von Dingen, die auf dem Zielrechner (dem Node) konfiguriert werden sollen. Resources selbst besitzen keine Ausführungslogik. Zu jeder Resource gibt es aber einen oder mehrere Provider, die für eine oder mehrere Plattformen die tatsächliche Ausführung übernehmen. Tatsächlich funktioniert das in der Regel nur für die gängigen Linux-Distributionen.

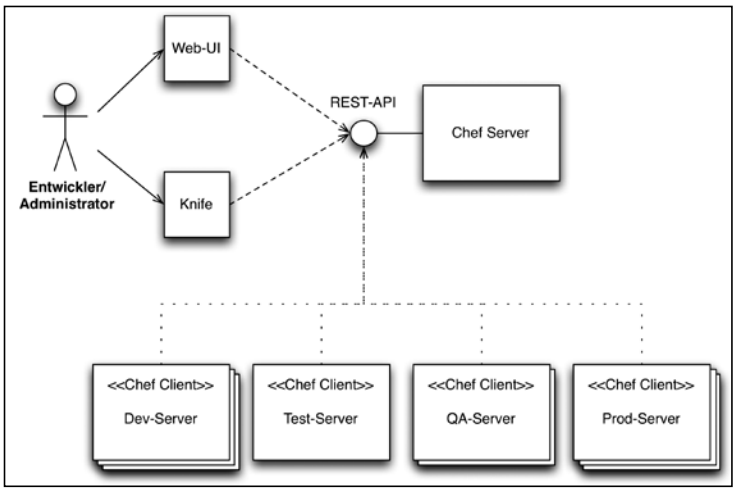


Abb. 1: Chef Client/Server

Chef unter Windows sei vorerst nur den experimentierfreudigeren Menschen empfohlen. Betrachten wir als Beispiel die Package Resource (Listing 1). Sie kann genutzt werden, um benötigte Softwarepakete zu installieren. Da es aber viele verschiedene Wege gibt, Pakete zu installieren, existieren auch etliche verschiedene Provider für diese Resource: darunter für Apt, Rpm, MacPorts oder Yum. Durch Angabe des *Provider*-Attributs kann ein bestimmter Provider gewählt werden, ansonsten wird abhängig vom Zielsystem ein sinnvoller Default gewählt.

Ein weiteres Attribut, das jede Resource besitzt, ist *action*. Es bestimmt, welche Aktion ausgeführt werden soll. Die gültigen Werte sind abhängig von der konkreten Resource; für die Package Resource sind es *install*, *upgrade*, *remove* und *purge*. Die meisten verfügbaren Resources sind idempotent. Das heißt, sie können mehrfach Ausgeführt werden, ohne dass es zu einem Fehler kommt. Zum Beispiel wird ein Provider für die Package Resource beim Ausführen der Aktion *install* zunächst prüfen, ob das geforderte Paket schon installiert ist. Nur wenn das nicht der Fall ist, wird eine Installation durchgeführt. Falls die Installation aus temporären Gründen scheitert (weil z. B. ein Software-Repository nicht verfügbar ist), wird sie beim nächsten Lauf des Chef-Clients erneut gestartet. Auf diese Weise konvergiert die tatsächliche Konfiguration des Zielsystems gegen das gewünschte Ergebnis. Einige Resources sind nicht von sich aus idempotent, hier muss der Entwickler selbst sicherstellen, dass es zu keinem Fehler kommt, wenn sie mehrfach ausgeführt werden.

Listing 1: Package Resource

```
package "tar" do
  # Alternativen: upgrade, remove, purge
  action :install
  # Wahl eines Providers statt des Defaults
  provider Chef::Provider::Package::Rpm
end
```

Das gilt zum Beispiel für die Script Resource, mit der vorhandene Bash-, Perl-, Python-, Ruby- oder CSH-Skripte ausgeführt werden können.

Template Resource

Eine häufig wiederkehrende Aufgabe ist das Anlegen von Konfigurationsdateien. Chef unterstützt das durch die Template Resource. Dazu wird im Cookbook ein Template-File angelegt, auf das dann im Rezept verwiesen wird. Die Template-Dateien werden dabei im *.erb*-Format abgelegt, sozusagen in der Ruby-Version des JSP-Markups. Dabei kann in Tags der Form <% *some_Code* %> Ruby-Code eingebettet werden. Ein Platzhalter der Form <%= *expression* %> wird durch den Wert des Ausdrucks ersetzt. Listings 2 und 3 zeigen, wie Platzhalter in einem Template verwendet werden können und wie im Recipe Platzhalter an Werte gebunden werden (hier wird @*Name* an „Welt“ gebunden).

Die Template Resource verfügt (ebenso wie einige verwandte Resources) über einen praktischen Mechanismus, um Cookbooks für mehrere Plattformen fit zu machen: Im *template*-Ordner des Cookbooks kann eine Datei des gleichen Namens in verschiedenen Unterordnern abgelegt werden. Chef wählt dann anhand des Hostnamens oder des Betriebssystems die passende Version aus. Für unser Beispiel könnte das folgendermaßen aussehen:

```
templates
    eigenbrodt.example.org/beispiel.erb
    ubuntu-11.04/beispiel.erb
    ubuntu/beispiel.erb
    default/beispiel.erb
```

Listing 2: beispiel.erb

```
<body>
  Hallo <%=@Name%>. Ich zähle bis 10:
  <ul>
    <%  (1..10).each {|i|  %>
      <li> <%=i%> </li>
    <% } %>
  </ul>
</body>
```

Listing 3: Template Resource

```
# Es wird eine Datei mit dem Namen /tmp/beispiel.html
template "/tmp/beispiel.html" do
  source "beispiel.erb"
  variables(
    :Name => "Welt"
  )
end
```

Knoten und Attribute

Bei den bisher vorgestellten Beispielen handelt es sich im Grunde um „dumme“ Skripte. Sie unterscheiden sich prinzipiell nicht von anderen Lösungen wie Shell- oder Perl-Skripten. Sie können in der gezeigten Form auf einem Rechner mit Chef Solo auch ähnlich verwendet werden. Seine eigentliche Stärke entfaltet Chef aber beim Einsatz eines Chef-Servers. Der Chef-Server fügt einige wichtige Konzepte hinzu. Seine wichtigste Fähigkeit ist, Informationen über die verwalteten Server zu sammeln und zu speichern. Ein Server wird in Chef durch einen Node repräsentiert. Wenn ein neuer Server konfiguriert werden soll, muss er Chef als Node bekannt gemacht werden. Welche Nodes es gibt, kann mit *knife* abgefragt werden:

```
$ knife node list
testServer
testServerZwei
```

Mit *knife* können wir auch noch mehr Informationen über einen speziellen Knoten abrufen:

```
$ knife node show testServer
Node Name:      testServer
Environment:    _default
FQDN:           ip-10-56-66-177.eu-west-1.compute.internal
IP:             46.137.49.126
Run List:       recipe[exampleRecipe]
Roles:
Recipes:        exampleRecipe
Platform:       ubuntu 11.04
```

Bei den angezeigten Daten handelt es sich um eine sehr kleine Auswahl von wichtigen Attributen des Node. Mit der Opption *-l* können alle Attribute angezeigt werden. Chef (genauer ein Tool namens Ohai, das auf dem entfernten Rechner ausgeführt wird) sammelt von sich aus zahlreiche nützliche Informationen und speichert sie als Attribute des Knotens. Darunter zum Beispiel IP-Adresse, Hostname, Informationen über installierte Programmiersprachen, die Prozessorarchitektur, die Hauptspeichergröße und Netzwerkschnittstellen.

Im obigen Beispiel sehen wir, dass der Knoten außerdem über eine Run List verfügt. Sie bestimmt, welche Recipes und Roles (Rollen) auf dem Knoten ausgeführt werden sollen. Roles sind eine Abstraktion, mit der sich mehrere Recipes (und Attributes) zusammenfassen lassen. So könnte eine Role *LAMP* zum Beispiel aus den Recipes *Apache*, *Mysql* und *PHP* bestehen. Roles vereinfachen die Verwaltung mehrerer gleichartiger Server. Der Run List eines Knotens muss dann nur die Rolle *LAMP* zugewiesen werden, um alle drei Rezepte darauf auszuführen. Mit *knife* können Knoten Recipes oder Roles zugewiesen oder entfernt werden. Zum Beispiel kann mit folgendem Code dem Knoten *testServer* das Recipe *tomcat* hinzugefügt werden: *knife node run_list add testServer recipe[tomcat]*. Neben den automatischen Attributen können Knoten beliebige eigene At-

tribute hinzugefügt werden. Damit lassen sich Recipes parametrisieren. Es gibt vier verschiedene Ebenen von Attributen (in aufsteigender Reihenfolge):

- *default*
- *normal*
- *override*
- *automatic*

Automatic Attributes können vom Benutzer nicht verändert werden, es sind jene, die durch Ohai gesetzt werden. Alle anderen können durch Attribute-Dateien in Cookbooks, durch Roles oder direkt am Node gesetzt werden. Bei mehrfacher Setzung des gleichen Attributs gilt eine festgelegte Reihenfolge:

- *default attributes* in Attributdateien
- *default attributes* in Roles
- *default attributes*, die auf einem Node gesetzt sind
- *normal attributes* in Attributdateien
- *normal attributes*, die auf einem Node gesetzt sind
- *override attributes* in Attributdateien
- *override attributes* in Roles
- *override attributes*, die auf einem Node gesetzt sind

Üblicherweise setzen Cookbooks Default-Werte in ihren Attributdateien (Listing 4). In Rollen oder auf konkreten Knoten werden sie dann gegebenenfalls überschrieben. Innerhalb von Recipes kann über *node* auf die Attribute zugegriffen werden: *node[:http_port]*.

Databags

node-Attribute gelten immer im Kontext eines Servers. Es gibt aber auch die Möglichkeit, globale Parameter geordnet abzulegen. Das geschieht in so genannten Databags. In Databags können beliebige JSON-Daten global gespeichert werden. Außerdem können sie in Recipes ausgewertet werden. Angelegt werden sie typischerweise mit *knife*. Der folgende Befehl erstellt eine Sammlung von Benutzern: *knife data bag create benutzer*. Ein einzelner Benutzer kann nun zum Beispiel mit *knife data bag create benutzer peter* angelegt werden. Entsprechend konfiguriert öffnet *knife* nun einen Texteditor, in dem die JSON-Struktur für diesen Benutzer eingetragen werden kann. Das *id*-Attribut ist dabei schon vorbelegt, alles andere ist frei wählbar, zum Beispiel:

```
{
 "id": "peter",
 "public_key": "ssh-rsa AAAAB3NzaC1yc2EAA...."
}
```

Listing 4: attributes/attribute.rb

```
node.default["http_port"]  = "80"
# Node muss nicht unbedingt angegeben werden
default["ssh_port"]  = "22"
```

Nach dem Schließen des Editors werden die Daten an den Chef-Server übertragen. Innerhalb von Recipes kann nun auf diese Daten ähnlich wie auf *node*-Attribute zugegriffen werden: *peter = data_bag_item('benutzer', 'peter')*.

Suchen

Attribute und Databags werden auf dem Server persistiert. Dadurch ist es möglich, nach Nodes oder Databags zu suchen. Die Suchen können mit *knife* aber auch innerhalb von Recipes durchgeführt werden. Zum Beispiel liefert *knife search node 'name:test*' eine Liste aller Knoten, deren Name mit „test“ beginnt. Suchen sind nützlich, um Beziehungen zwischen verschiedenen Servern dynamisch zu verwalten. So kann zum Beispiel ein Webserver dynamisch eine Liste aller konfigurierten Application-Server erfragen und seine Lastverteilung entsprechend konfigurieren. Ebenso könnte ein Recipe die öffentlichen Schlüssel einer Reihe von Benutzern ermitteln und ihnen SSH-Zugang zum konfigurierten Rechner gewähren.

Teilen und Erweitern

Durch die Parametrisierbarkeit und das vergleichsweise hohe Abstraktionsniveau der DSL entsteht die Möglichkeit, wiederverwendbare Cookbooks zu schreiben. Tatsächlich existieren bei opscode [2] und auf GitHub schon zahlreiche Cookbooks für verschiedenste Anwendungsfälle. Aber auch die Sprache selbst ist erweiterbar. So ist es problemlos möglich, eigene Resource-Typen zu definieren, und bei Recipes handelt es sich um reine Ruby-Programme, in denen praktisch alles möglich ist. Chef ist also nicht einfach ein Tool, sondern ein Framework, das an die eigenen Bedürfnisse angepasst werden kann.

Ausblick

Dieser Artikel hat die grundlegenden Konzepte von Chef beleuchtet und gezeigt, dass Infrastructure as Code ein vielversprechender Ansatz ist. Allerdings soll nicht verschwiegen werden, dass er auch Risiken birgt: Der hohe Grad der Automatisierung erfordert auch automatisierte Tests, auf die hier nicht eingegangen wurde. Noch hat sich kein standardisiertes Vorgehen hierfür etabliert und auch die meisten öffentlich verfügbaren Cookbooks enthalten keine Tests. Als Einstieg ins Thema Testen können die Links [3] und [4] dienen.

Martin Eigenbrodt ist Senior Consultant bei der innoQ Deutschland GmbH. Neben der Softwareentwicklung mit JavaEE beschäftigt er sich mit Build, Deployment und DevOps-Themen.

Links & Literatur

[1] http://www.opscode.com/chef/

[2] http://community.opscode.com/cookbooks

[3] http://www.cucumber-chef.org/

[4] http://vagrantup.com/

Was wir von Lean-Start-ups lernen können

Hype oder hilfreich?

DevOps – die engere Verbindung von Entwicklung (Development) und Betrieb (Operations) – ist ein Thema, das zurzeit heiß diskutiert wird. Aber was ist dran am Hype? Ist DevOps nur eine Modeerscheinung, gepusht von Agilitätsromantikern?

von Oliver Wolf

Wenn man die Blogs der Technikteams großer Websites wie Wordpress.com, Flickr, Etsy und anderer verfolgt, möchte man seinen Augen kaum trauen: Von bis zu fünfzig Deployments in die Livesysteme pro Tag ist da die Rede, von fünfundzwanzigtausend Releases in etwas über vier Jahren und so weiter und so fort [1], [2]. Ich weiß nicht, wie es Ihnen geht, aber ich fühle mich dabei schlecht. Ich erinnere mich schmerzlich zurück an betriebliche Abnahmen, die nach Monaten daran gescheitert sind, dass noch irgendeine Betriebsdokumentation gefehlt hat. Sie scheiterten auch an Staging-Phasen, die oft länger gedauert haben als die eigentliche Entwicklung. Und ich frage mich: Was um alles in der Welt machen die eigentlich anders und vielleicht auch besser?

Zeitreise

Um das zu verstehen, reisen wir ein wenig in der Zeit zurück, ungefähr in das Jahr 2004/2005. Der IT-Markt hatte nach dem Zusammenbruch der DotCom-Blase im Frühjahr 2000 die Talsohle durchschritten und eine neue Gründergeneration begann vor allem in den Vereinigten Staaten den Grundstein für das zu legen, was man „Web 2.0“ nennt. Die Aufbruchsstimmung war ähnlich wie Ende der 1990er Jahre, aber eines war grundlegend anders: Investorengelder gab es diesmal nicht mehr für Visionen, sondern nur noch für laufende Software. Für Start-up-Unternehmer zu dieser Zeit bedeutete das, möglichst schnell zu geringstmöglichen Kosten ein erstes vorzeigbares Produkt auf den Markt zu bringen und es dann kontinuierlich zu erweitern und zu verbessern, sobald (und falls) das Geld anfing zu fließen. Die Revolution brachten dynamische Programmiersprachen wie Ruby und Webframeworks wie Rails, mit denen Applikationsentwicklung plötzlich so effizient möglich wurde wie nie zuvor (und deren Konzepte nebenbei bemerkt mittlerweile auch die Java-Welt maßgeblich beeinflusst haben). Und auch die Betriebsinfrastruktur wurde einfacher: Keine schwerfälligen Applikationsserver mehr, für deren Konfiguration und Betrieb Spezialisten und leistungsfähige Hardware gebraucht wurden, sondern leichtgewichtige Laufzeitumgebungen, die sich mit virtuellen Maschinen beim Hoster des Vertrauens begnügten. Fand ein Kunde ein Problem, wurde der Fehler behoben und eine neue Version in Produktion gebracht – das Ganze oft innerhalb von wenigen Stunden oder gar Minuten. Die Teams waren klein und motiviert, und für Entwicklung und Betrieb war meist ein und dieselbe Person zuständig. „Lean-Start-up“ nennt sich das mittlerweile, und es steht für nicht weniger als einen massiven Paradigmenwechsel.

„Die Realität ist in Wirklichkeit ganz anders!“ (Helmut Kohl)

Zurück in der klassischen Corporate-IT-Welt des Jahres 2011 ist davon noch nicht viel zu spüren. Agile Methoden haben sich zwar in der Softwareentwicklung inzwischen auf breiter Front durchgesetzt, und selbst in Traditionshäusern gehört es neuerdings zum guten Ton, dass man selbstverständlich „Scrum macht“, man ist ja schließlich nicht von gestern. Spätestens dort aber, wo Unternehmenssoftware dummerweise erst wirklich anfängt, einen Beitrag zur Wertschöpfung zu leisten, ist meist das Ende der agilen Fahnenstange erreicht. Unzählige Entwickler, Architekten und Projektleiter sind schon schier daran verzweifelt, das Ergebnis wochen- und monatelanger Mühe endlich über die Schwelle der heiligen Hallen des Rechenzentrums zu tragen. „Njet“, bekommen sie zu hören, „so geht das nicht“, und im Augenwinkel knallt noch der Schuh auf den Schreibtisch. Alles Schikane? Ziemlich sicher nicht, wenn wir mal die Perspektive wechseln und die ganze Sache aus der Sicht eines Systemadministrators sehen. Stellen Sie sich vor, wie es ist, wenn Sie um drei Uhr morgens aus dem Bett geklingelt werden und versuchen müssen, ein geschäftskritisches System wieder ans Laufen zu bekommen, über das Sie kaum etwas wissen. Sie haben einen Waschzettel mit ein paar Hinweisen zum Starten und Stoppen und wissen bestenfalls ungefähr, wo die Log-Dateien liegen. Zu dumm, dass sich darin außer ein paar Java Exceptions kein Hinweis auf den Fehler findet – und von Java haben Sie als Unix-Experte ungefähr so viel Ahnung wie ein Schwein vom Klettern. Und um diese Zeit einen Entwickler ans Telefon zu bekommen, ist illusorisch, und das System ist nur eines von unge-

fähr fünfzig, die Sie und Ihre Kollegen betreuen müssen. Viel Spaß!

Sie sehen, worauf ich hinaus will; das Dilemma ist offensichtlich: Die Innovationszyklen in unserer Branche werden immer kürzer, der Markt ruft nach neuen Produkten, der Wettbewerb schläft nicht, wer rastet, der rostet. Die Fähigkeit zur ständigen Veränderung ist heute mehr denn je ein kritischer Erfolgsfaktor. Auf der anderen Seite: Ausfälle und Pannen haben in den allermeisten Fällen ihre Ursache in irgendeiner Veränderung an Bestehendem. Da ist es nur verständlich, dass im klassischen Systembetrieb Veränderung eher als Ausnahme gesehen wird und Prozessframeworks wie ITIL die Komplexität von Änderungsprozessen auch nicht gerade reduzieren. „If it ain't broke, don't fix it." – ich glaube, es gibt Administratoren, die das sogar als Aufdruck auf ihrem T-Shirt tragen.

Wenn Veränderung nun also für Unternehmen notwendig ist, um wettbewerbsfähig zu bleiben, gleichzeitig aber damit die Gefahr einhergeht, dass etwas schief läuft, gibt es eigentlich nur eine Lösung: notwendige und sinnvolle Veränderungen zulassen und gleichzeitig Mittel und Wege finden, das Risiko von Fehlern zu reduzieren. Und damit sind wir genau bei der Essenz dessen angekommen, was Unternehmen mit traditioneller Aufgabenteilung in der IT von Lean-Start-ups lernen können.

Kultur ...

Zwei Dinge braucht es nämlich dazu: eine passende Unternehmenskultur und die geeigneten Praktiken und Werkzeuge. Der erste Teil, die Unternehmenskultur, ist der einfachste und der schwierigste zugleich. Der einfachste ist er, weil es im Kern nur darum geht, gemeinsam an einem Strang zu ziehen und zu verstehen und zu akzeptieren, dass beide Rollen – Entwicklung und Betrieb – ihren Beitrag zur Wertschöpfung eines Unternehmens zu leisten haben. Ein Rechner, auf dem nichts läuft, ist nun einmal nicht viel mehr als wertloses Blech. Und eine Software, die nicht betrieben wird, ist nur eine akademische Fingerübung. Der schwierigste ist er, weil die Realität leider meistens anders aussieht. Betriebsorganisationen werden eher an Uptime und Fehlerrate gemessen als zum Beispiel an der Flexibilität der Deployment-Prozesse – und für Entwickler und Projektleiter zählt die Anzahl der termin- und budgetgerecht umgesetzten Features nicht primär die Betriebsfreundlichkeit der Software. Dabei hilft es oft, rechtzeitig miteinander zu reden: Die Fehlermeldung „Configuration file foo.cfg not found at file system location /etc/foo" etwa hilft einem Administrator einfach deutlich weiter als „Component foo.bar.Foo not configured". Dennoch: kulturelle Änderungen brauchen Zeit, Energie und letztendlich auch die Unterstüzung „von oben", wenn sie Früchte tragen sollen – nichts also, was man schnell erreichen kann. Bleiben noch die Werkzeuge.

... und Werkzeuge

Ein wichtiges Werkzeug für die Softwareentwicklung ist heute „Continuous Integration", kurz CI. Nach dem Motto „Fail early, fail fast, fail often" geht es hier darum, dem Entwickler eine sehr zeitnahe und unmittelbare Rückmeldung über die Auswirkungen einer von ihm durchgeführten Änderung zu geben. Voraussetzung dafür sind umfangreiche Tests, die automatisiert und schnell ablaufen können und eine zuverlässige Aussage über den Zustand des Systems geben müssen. Führt man den Gedanken ein wenig weiter und schließt auch das automatische Deployment in eine Laufzeitumgebung mit ein, landet man bei „Continuous Deployment/Delivery" (CD), einer der Schlüsselpraktiken, die Lean-Start-ups so schnell machen. Viele Minireleases mit kleinen Änderungen sind meist weniger risikoreich als große Releases mit vielen Änderungen, so die Überlegung, und zudem wesentlich leichter rückgängig zu machen. Auch wenn die Laufzeitumgebung letztendlich noch nicht die „echte" Produktion ist, sondern nur ein QA- oder Pre-Production-System, gewinnt man durch Continuous Deployment viel Flexibilität und Sicherheit und kann auf lange Sicht mit der Qualität der häufig gelieferten Releases vielleicht auch langsam den Betrieb überzeugen, dass dahinter mehr steckt als eine kurzlebige Modeerscheinung.

Continuous Deployment ist nur eines von vielen Beispielen für Praktiken und Werkzeuge. Allen gemeinsam ist aber eines: der starke Fokus auf der Automatisierung von ansonsten komplexen, fehleranfälligen Prozessen, die durch eine neue Klasse von leistungsfähigen und flexiblen Frameworks unterstützt wird. Ein mittlerweile recht bekannter Vertreter aus dieser Kategorie ist „Chef", zu dem Sie in dieser Ausgabe einen ausführlicheren Artikel (auf Seite 53) lesen können.

Fazit

Warum erzähle ich Ihnen das alles? Ganz einfach: Weil ich glaube, dass auch klassisch aufgestellte Unternehmen von einer stärkeren Verbindung zwischen Entwicklung und Betrieb profitieren können. Wahrscheinlich ist Ihr Ziel nicht, fünfzig produktive Deployments am Tag zu erreichen. Vielleicht möchten Sie nur die Zeit bis zur Produktivschaltung eines neuen Releases von drei Monaten auf einen Monat reduzieren. Aber auch dann lohnt es sich, einen Blick in den Werkzeugkasten der DevOps zu werfen. Und wer weiß, vielleicht hat das ja am Ende sogar noch positive Auswirkungen auf die Kultur?

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

Links & Literatur

[1] http://bit.ly/u4JimV

[2] http://bit.ly/sU0tz5

innoQ Briefing: IT-Trends

Bringen Sie Licht in den Dschungel

In immer rasanteren Abständen kommen in der IT neue Technologien und Paradigmen auf den Markt. Wann bleibt im Arbeitsalltag überhaupt noch Zeit, innezuhalten und sich mit technischen Neuerungen zu beschäftigen?

Sich einen guten Überblick zu verschaffen und vor allem das für die eigene Arbeit Relevante herauszufiltern, also die Ansätze zu finden, die zur Unternehmensstruktur passen und eine positive Entwicklung unterstützen, ist für IT-Entscheider und -Spezialisten heute schwieriger denn je.

Wir lösen das – persönlich!

www.innoQ.com

innoQ Deutschland GmbH · info@innoq.com · Tel. +49 2102 77162-100
innoQ Schweiz GmbH · info@innoq.com · Tel. +41 41 743 01-11