Kai Weingärtner
(kai.weingaertner@opitz-consulting.com)
ist als Senior Consultant bei der OPITZ CONSULTING Deutschland GmbH am Standort Hamburg tätig. Neben seiner über zehnjährigen Erfahrung in der Entwicklung und Konzeption von Java-Enterprise-Anwendungen interessiert er sich heute besonders für Fragen der Qualitätssteigerung im Softwareentwicklungsprozess. Darüber hinaus berät er Kunden in den Bereichen serviceorientierte Architekturen und Geschäftsprozessmanagement.

# DevOps Getting Started – Wie der Einstieg gelingt

Wenn man Artikel über DevOps und Continuous Delivery liest, werden oft Firmen mit mehreren täglichen Deployments wie „Netflix“ oder „Etsy“ angeführt. Der Alltag sieht in den meisten Unternehmen allerdings ganz anders aus, sodass man dazu verleitet wird, das Thema DevOps „ad acta“ zu legen. Die Ziele scheinen einfach zu weit entfernt. Dabei wird leicht übersehen, dass auch Netflix oder Etsy etliche Iterationen durchlaufen haben, ehe sie zu ihrem jetzigen Vorgehen gelangt sind. Jeder startet also mit einem ersten Schritt. Mit diesem Artikel möchten wir Sie dazu anregen, sich die Praktiken von DevOps schrittweise zu eigen zu machen. Anhand eines beispielhaften Verlaufs zeigen wir Ihnen, wie Sie schnell erste Erfolge erzielen, ohne gleich eine Kulturrevolution auszurufen.

## Ziele von DevOps

DevOps steht für eine ganzheitliche Optimierung der Lieferkette vom “CodeCommit“ bis zur ausgelieferten oder lieferbereiten Anwendung. Wichtig ist es dabei, dass Sie folgende Aspekte von Anfang an berücksichtigen:

- Um den gesamten Lieferprozess in die Betrachtung einzubeziehen, müssen Entwicklung und Betrieb zusammenarbeiten. Lokale Optimierungen sollten Sie vermeiden, diese würden die bestehenden „Flaschenhälse“ lediglich verlagern.
- Um bei der gewünschten hohen Geschwindigkeit dennoch die nötige Stabilität zu gewährleisten, sollten Sie Varianzen durch manuelle Eingriffe in den Prozess vermeiden. Automatisierung muss daher ein zentrales Ziel sein.
- Die Automatisierung darf das Vertrauen in die Lieferung nicht beeinträchtigen. Sorgen Sie deshalb für ausreichende Feedback-Kanäle und Prüfschritten in der Lieferkette.
- Zuletzt muss jede Änderung in der Lieferkette, sei es der Anwendungscode, die Infrastruktur oder der Lieferprozess selbst, nachvollziehbar bleiben. Dies erfordert eine durchgängige Versionierung aller Quellen und Build-Erzeugnisse.

## Wo fangen wir an?

Starten Sie am besten mit einem kleinen Team, das zusammen den gesamten Lieferprozess technisch und organisatorisch im Detail kennt. Dies sollten Mitarbeiter aus der Entwicklung und dem Betrieb sein, die bedarfsweise temporär noch durch Build- und Release-Manager unterstützt werden. Die festen Teammitglieder fungieren später auch als Wissensmultiplikatoren in ihren Teams. Für den Start ist es außerdem sinnvoll, einen DevOps-Engineer einzubinden, der sich mit den Automatisierungswerkzeugen auskennt und Erfahrung mit DevOps-Initiativen hat.

In vielen Unternehmen besteht ein Release aus mehreren Anwendungen, die koordiniert ausgeliefert werden müssen. Zum Start empfiehlt es sich, eine einzelne Anwendung zu wählen, die autark bis in Produktion geliefert wird, aber trotzdem eine gute Sichtbarkeit im Unternehmen hat, damit Sie mit ihr für das DevOps-Vorgehen werben können.

Zu Beginn wird der Lieferprozess, also der Weg, den ein Commit vom Check-in bis in Produktion nimmt, im Detail erfasst. Dabei hilft Ihnen die Technik des Value-Stream-Mappings [VSM].

Anschließend lassen sich die Stellen identifizieren, an denen manuell in den Prozess eingegriffen wird, Varianzen hereinkommen und Koordinationsaufwand entstehen.

Sehr oft findet man:

- Manuelle Arbeiten bei der Bereitstellung der Integrations- und Testumgebungen sowie beim Einspielen späterer Konfigurationsänderungen.

- Manuelle Arbeiten bei der Migration von Datenbankänderungen und der Bereitstellung von Testdaten.
- Koordinationsaufwand bei der Übergabe von einer Umgebung in die nächste. *Beispiel*: Für das Einspielen einer Datenmodelländerung muss das Datenbankteam beauftragt werden, oder beim Einspielen eines Anwendungsstands in die Testumgebung muss die Entwicklung beauftragt werden.

Gerade der erste Punkt bietet viel Optimierungspotenzial, da er häufig durchgeführt wird und die Nachvollziehbarkeit von Änderungen essenziell für die Stabilität und die Qualität der Umgebungen ist.

## Automatisierte Bereitstellung der Infrastruktur

Bei der automatisierten Bereitstellung einer Anwendungsumgebung helfen Konfigurationsmanagementwerkzeuge wie Puppet, Saltstack, Chef und Ansible. Diese eignen sich gut zum Einstieg in die Automatisierung, weil sie sich leicht in bestehende Infrastrukturen einbinden lassen und ein schrittweises Vorgehen bei der Automatisierung des Umgebungssetups ermöglichen. Am Beispiel Ansible möchten wir Ihnen zeigen, wie Sie dabei vorgehen können.

Eine Systemdefinition wird in Ansible in einem Playbook beschrieben. Listing 1 zeigt ein einfaches Playbook, das wir später noch näher erläutern. Dieses führen Sie mit dem Befehl ansible-playbook aus. Die Tasks in dieser YAML-Datei werden sequentiell durchlaufen.

Jeder Task gibt einen Zielzustand an und wird nur ausgeführt, wenn dieser noch nicht erreicht ist. Jedes der bereits genannten Tools bietet diese „Idempotenz“ und ermöglicht damit eine Ausführung, obwohl Teile der Anwendung schon in der Umgebung vorhanden sind.

Für das iterative Vorgehen ziehen Sie nun eine vollständig konfigurierte Umgebung heran und nehmen nacheinander alle Änderungen gegenüber dem Initialsystem in das Playbook auf. Nach jeder Änderung sollte mit einem Dryrun (Parameter-check) sichergestellt werden, dass die Ausführung keine Änderung bewirkt und die Definition tatsächlich dem aktuellen Zustand entspricht.

Um reproduzierbare Ergebnisse zu erhalten, legen Sie jedes Softwarepaket, das

Listing 1

```
---
- hosts: appservers

 tasks:
 - name: JDK Installieren
  yum:
    name: http://rpmrepo/jdk-pgk_1.8.20.rpm
    state: present

 - name: Application Server installieren
  yum:
    name: http://rpmrepo/wildfly-pgk_9.0.1.rpm
    state: present

 - name: Application Server konfigurieren
  template:
   src: templates/appserver/standalone.xml.j2
   dest: /opt/wildfly/standalone/configuration/standalone.xml
  notify: Appserver restart

 handlers:
 - name: Appserver restart
  service:
    name: wildfly
    state: restarted
```

über das Playbook installiert wird, in einer definierten Version im Software-Repository ab. Alle einmal verwendeten Versionen müssen dort dauerhaft aufbewahrt werden. In Listing 1 haben wir für JDK und Wildfly RPM-Pakete erstellt. Repositorys wie Artifactory oder Nexus können als Ablage für solche RPM-Pakete dienen. RPM-Pakete erstellen Sie z. B. mit dem Tool FPM [FPM] aus verschiedenen Eingangsformaten wie Tarballs.

Müssen Konfigurationsdateien angepasst werden, können Sie wie in Listing 1 bei der Wildfly-Konfigurationsdatei standalone.xml Templates verwenden. Dazu kopieren Sie die Datei und ersetzen veränderliche Teile, wie z. B. Variablen, durch Template-Ausdrücke. Diese Templates werden mit dem Playbook abgelegt. Bei der Ausführung werden dann die Variablenwerte ersetzt. Dies wird für die umgebungsspezifischen Teile der Konfiguration wichtig.

Das Playbook ist vollständig, wenn alle Infrastruktur-Softwarepakete und Konfigurationseinstellungen aufgenommen wurden, die nötig sind, um ein Deployment gegen das System durchzuführen. Abschließend sollte das Playbook gegen ein initiales System ausgeführt und das Ergebnis mit dem vorkonfigurierten System verglichen werden. Dazu können Sie z. B. beide Dateisysteme mit rdiff vergleichen.

Wenn das Playbook funktioniert, ist der nächste Schritt, alle umgebungsspezifischen Konfigurationswerte zu identifizieren und auszulagern, damit Sie das Playbook später einheitlich in allen Umgebungen nutzen können. Bei Ansible eignet sich dafür z. B. die Inventory-Datei, die gleichzeitig der Host-Gruppe im Playbook konkrete Servernamen zuordnet (siehe Listing 2). Die für die Umgebung passende Inventory-Datei können Sie beim Playbook-Aufruf mitgeben.

Listing 2

```
---
[appservers]
appsrv1
appsrv2

[appservers:vars]
db-endpoint=jdbc:oracle:thin:@dev-db:1521:APP"
db-user=username
db-pass=password
```

Allerspätestens jetzt sollte das Playbook mit allen Konfigurationsdateien unter Versionskontrolle gestellt werden. Alle weiteren Aktualisierungen von Umgebungen sollten danach immer dem gleichen Schema folgen:

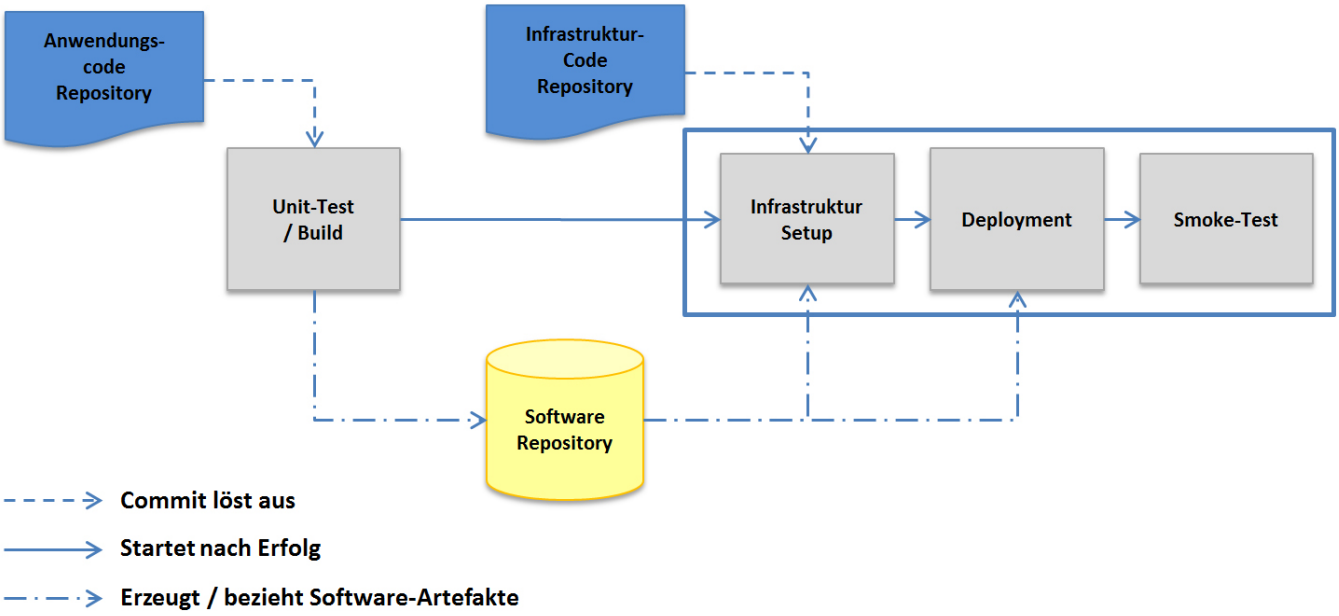


*Abb.: Einbindung in die Build-Pipeline*

1. Anpassen der Systemdefinition
2. Test in einer Testumgebung
3. Check-in
4. Automatisches Auschecken und Ausführen der aktuellen Systemdefinition durch das Konfigurationsmanagementwerkzeug.

## Infrastructure as Code

Sobald die Systemdefinition effektiv in Code-Form vorliegt, lässt sich dieser Code nach den gleichen Coding-Praktiken weiterentwickeln, die für Anwendungscode gelten:

- Modularisierung: Ansible Roles können genutzt werden, um zusammenhängende Konfigurationsaspekte zu modularisieren. Andere Konfigurationsmanagementwerkzeuge bieten vergleichbare Konzepte.
- Peer Reviews oder Pull Requests können genutzt werden, um die Qualität einer Änderung vor dem Commit zu prüfen.
- Ganz wichtig: Tests
  Dazu gehört:
  - Änderungen vor dem Commit in einer Testumgebung validieren,
  - mit Prüfschritten (asserts) im Playbook die Vor- und Nachbedingungen sicherstellen,
  - Testen der Idempotenz einzelner Tasks.

  Wer Ruby-Entwicklung nicht scheut, kann mit Serverspec sogar eine Funktionstest-Suite für das Umgebungssetup schreiben [ServerSpec].

Entwickler sind mit diesen Coding-Praktiken vertraut und sollten sie im DevOps-Team weitervermitteln.

## Deployment

Mit etwas Glück existiert in Ihrem Unternehmen bereits eine Lösung für ein automatisches Deployment. Vermeiden sollten Sie jedoch in jedem Fall unterschiedliche Deployment-Verfahren für die verschiedenen Umgebungen. Um mit dem Durchreichen eines Releases durch die Umgebungen Vertrauen in den Deployment-Prozess zu gewinnen, ist es wichtig, ein einheitliches Deployment-Verfahren in allen Umgebungen durchzuhalten. Damit fungiert jedes Deployment als zusätzlicher Qualitätscheck des Verfahrens. Falls ein einheitliches Verfahren fehlt, können Sie auch hier Ansible nutzen, um das Deployment durchzuführen und umgebungsspezifische Parameter auszulagern.

Als abschließenden Deployment-Schritt richten Sie einen Smoke-Test ein, um frühzeitig Feedback über ein fehlerhaftes Deployment zu erhalten. Statt einen Fehler manuell in der Umgebung zu beheben, starten Sie den Lieferprozess nach der Änderung des Infrastruktur-Codes erneut.

Die Abbildung veranschaulicht die Einbindung der bisher genannten Schritte in eine Build-Pipeline. Die Schritte im Kasten wiederholen sich mit angepasster Umgebungskonfiguration für alle Zielumgebungen der Pipeline, wie der automatisierten Funktionstestumgebung, einer Fachtestumgebung oder einer Lasttestumgebung.

Auslöser der Pipeline können neben Anwendungs-Commits auch Änderungen des Infrastruktur-Codes sein. Damit stellen Sie sicher, dass die Anwendung auch bei reinen Infrastrukturänderungen mehrere Teststufen durchläuft, ehe sie produktiv geht.

Das tatsächliche Deployment in Produktion aus einem Commit heraus setzt ein hohes Maß an Vertrauen in die Qualität des Lieferprozesses voraus und sollte nicht das Ziel Ihrer ersten Iterationen sein. Vielmehr sollte darauf hingearbeitet werden, in der Lage zu sein, einen produktionsreifen Stand per Self-Service auf Knopfdruck ausliefern zu können.

## Self-Service

Sobald das Setup einer Umgebung und das Deployment in die Umgebung automatisiert sind, können diese theoretisch sofort erfolgen, sobald eine Anwendungsversion bereitsteht. Dies ist allerdings nicht immer praktikabel: Z. B. will ein Fachtester selbst entscheiden, wann er mit dem Test einer Version beginnt, oder der Betrieb möchte selbst bestimmen, wann ein Deployment in Produktion stattfindet.

Ohne automatisiertes Deployment entstehen hier Wartezeiten, da der Anforderer die Einspielung mit dem Lieferanten koordinieren muss. Mit automatisiertem Deployment hingegen können Sie einen Self-Service für den Anforderer einrichten, mit dem dieser per Knopfdruck den Zeitpunkt der Einspielung bestimmen kann, sobald eine Version vorliegt (Pull-Prinzip).

Build-Werkzeuge wie Go [Go] oder Bamboo [Bamboo], die das Build-Pipeline-Konzept integriert haben, bieten meist die Möglichkeit, eine Build-Stufe direkt manuell zu starten. In anderen wie Jenkins lässt sich dies über Plugins nachbilden z. B. mit dem Delivery-Pipeline-Plugin.

Falls das Build-Werkzeug hier keine Unterstützung bietet oder nicht freigegeben werden kann, stellen Sie z. B. über Rundeck [Run] eine Web-UI zum Starten von Deployments bereit, über die Sie das Ansible Playbook oder anderes ausführen. Um bei diesem Vorgehen eine Version zu kennzeichnen, die die manuelle Stage erfolgreich durchlaufen hat, bauen Sie ein separates Promoted-Build Repository auf, aus dem sich dann nachfolgende Build-Stufen bedienen.

## Immutable Infrastructure und virtuelle Umgebungen

Auch das beste Konfigurationsmanagementsystem kann die Integrität einer Umgebung nicht garantieren, wenn über ein Login auf dem System direkt Änderungen vorgenommen werden. Auch wenn die Änderungen den Weg über die Versionskontrolle nehmen, können Abweichungen zwischen Umgebungen entstehen, wenn diese nicht denselben Ausgangszustand haben. Bei langlebigen Umgebungen, die häufiger aktualisiert wurden, entsteht daher ein „Configuration Drift“, also Abweichungen zur eigentlich definierten Konfiguration.

Falls Sie mit virtuellen Umgebungen arbeiten, können Sie das Problem dadurch lösen, dass Sie bei jeder Änderung der Konfiguration ein neues, initiales Image bespielen und die alte Umgebung löschen. Falls ein Rollback erforderlich wird, würden Sie einfach die vorige Version der Systemdefinition einspielen. Falls die vollständige Bereitstellung einer neuen Umgebung aus dem Initialzustand zu lange dauert, erstellen Sie nach jeder Änderung ein Image und reichen dieses weiter.

Dafür können Sie z. B. Packer [Pac] in die Build-Pipeline einbinden und nach jedem Infrastruktur-“CodeCommit“ starten. Packer erstellt aus einem Basis-Image plus Infrastruktur-Code ein Image im benötigten Format. Mit der Definition in Listing 3 würde Packer ein VMware-Basis-Image mit einem Ansible Playbook modifizieren und das Ergebnis wieder als Image speichern.

Listing 3

```
---
{
 „builders“: [
  „type“: „vmware-vmx“,
   „source_path“: „basebox.vmx“,
   „ssh_username“: „root“
 ],
 „provisioners“: [
  „type“: „ansible-local“,
   „playbook_file“: „appserver-setup.yml“
 ]
}
```

## Produktionsnahe Umgebungen in der Entwicklung

Auch in der lokalen Entwicklung können Sie die einheitliche Infrastrukturdefinition nutzen, um so früh wie möglich das Verhalten in einer produktionsnahen Ablaufumgebung kennenzulernen. Dies erspart auch das lokale Einrichten der Umgebung, das sonst jeder Entwickler individuell vornehmen würde.

Das durch Packer erstellte Image kann lokal in eine Virtualisierungslösung wie VMware oder Virtualbox eingebunden und Shared Volumes für das direkte Deployment in die Box eingerichtet werden. Auch dies lässt sich zu einem Einzeiler automatisieren, wenn man Vagrant [Vag] verwendet.

Vagrant ist ein kommandozeilenbasiertes Werkzeug, das ein Image herunterlädt, welches in einem Vagrantfile beschrieben ist, und es wie dort beschrieben konfiguriert und startet. Für den Entwickler reduziert sich der Ablauf damit auf das Auschecken des Vagrantfiles (das im einfachsten Fall wie in Listing 4 aussieht) und auf das Ausführen des Befehls vagrant up.

Vagrant unterstützt sogar ein Multi-Machine-Setup, das mehrere VMs parallel startet. So kann eine Multi-Tier-Anwendung bestehend aus Web- und App-Server produktionsnah in lokalen VMs abgebildet werden.

Listing 4

```
---
VAGRANTFILE_API_VERSION = „2“
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
 config.vm.box = „appserver-1.0.0“
end
```

Vagrant kann auch als Umgebung für die Entwicklung des Infrastruktur-Codes genutzt werden, denn Vagrant bietet Unterstützung für alle gängigen Beschreibungsformate (siehe dazu [VagrantProv]). Falls angegeben, findet die Provisionierung direkt nach dem Starten der VM statt. Als Basis-Image verwenden Sie dann ein Image mit derselben Basisinstallation wie in den echten Umgebungen.

## Warum nicht gleich Container?

In letzter Zeit rücken Container-basierte Infrastrukturen und insbesondere Docker immer mehr in den Mittelpunkt der Betrachtung. Daher kann durchaus die Frage gestellt werden, ob man nicht im Rahmen der Infrastruktur-Automatisierung direkt auf Container umsteigen sollte. Neben einem geringeren Ressourcenverbrauch sind Container auch wie gemacht für den Immutable-Infrastructure-Ansatz, da sie noch schneller hochgefahren und wieder abgeräumt werden können als virtuelle Umgebungen. Wirklich interessant werden diese Vorteile in einer Microservices-Architektur, wo regelmäßig eine große Zahl an Umgebungen deployed wird.

Die vollständige Umstellung einer Anwendung auf Container birgt jedoch auch Risiken, aufgrund neuer Abläufe im Lieferprozess und neuem Verhalten im Betrieb. Daher sollten Sie im Vorfeld immer die Architektur dahingehend bewerten. Insbesondere bei komplexen Anwendungen halten wir die Automatisierung der bestehenden Abläufe für einen sinnvollen ersten Schritt, da diese Abläufe bereits erprobt sind.

## Fazit

DevOps ist nicht nur für Unternehmen wie Netflix und Etsy relevant. Jedes Unternehmen kann von den Prinzipien und Praktiken des Ansatzes profitieren. Ein kleines DevOps-Team und ein angemessenes Vorhaben, bieten den geeigneten Rahmen, um kulturelle Barrieren abbauen und notwendige technische Kenntnisse aufbauen zu können. Der dazu notwendige Erfahrungsaufbau kann schrittweise erfolgen, wobei schon Etappenziele Mehrwerte liefern.

In der Vereinheitlichung und Automatisierung der Umgebungserstellung und dem Deployment stecken häufig Optimierungspotenziale, die auch mit geringem Risiko zu heben sind. Werkzeuge wie An-

sible, Rundeck, Vagrant und Packer können dabei helfen, von individuell gepflegten Umgebungen zu einem einheitlichen und nachhaltigen Bereitstellungsprozess zu kommen, der sich in unterschiedlichen Umgebungen wiederholen lässt.

Die Infrastructure-as-Code-Praktiken und die produktionsnahe lokale Entwicklung zeigen, wie Entwicklung und Betrieb näher aneinanderrücken müssen damit DevOps in der Zusammenarbeit dieser Bereiche funktioniert. Also: Worauf warten Sie? Bilden Sie Ihr DevOps-Team und machen Sie den ersten Schritt! ■

## Quellen

[Ans] https://www.ansible.com/
[Bam] https://de.atlassian.com/software/bamboo
[FPM] https://github.com/jordansissel/fpm
[Go] https://www.go.cd/
[Pac] https://www.packer.io/
[Run] http://rundeck.org/
[Ser] http://serverspec.org/
[Vag] https://www.vagrantup.com/
[VagP] https://www.vagrantup.com/docs/provisioning/
[VSM] http://leanmanufacturingtools.org/551/creating-a-value-stream-map/