04-2016 | Winter | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Aus der Community – für die Community

Java aktuell

D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

Java ist vielseitig

JUnit 5
Das nächste große Release steht vor der Tür

Ansible
Konfigurationsmanagement auch für Entwickler

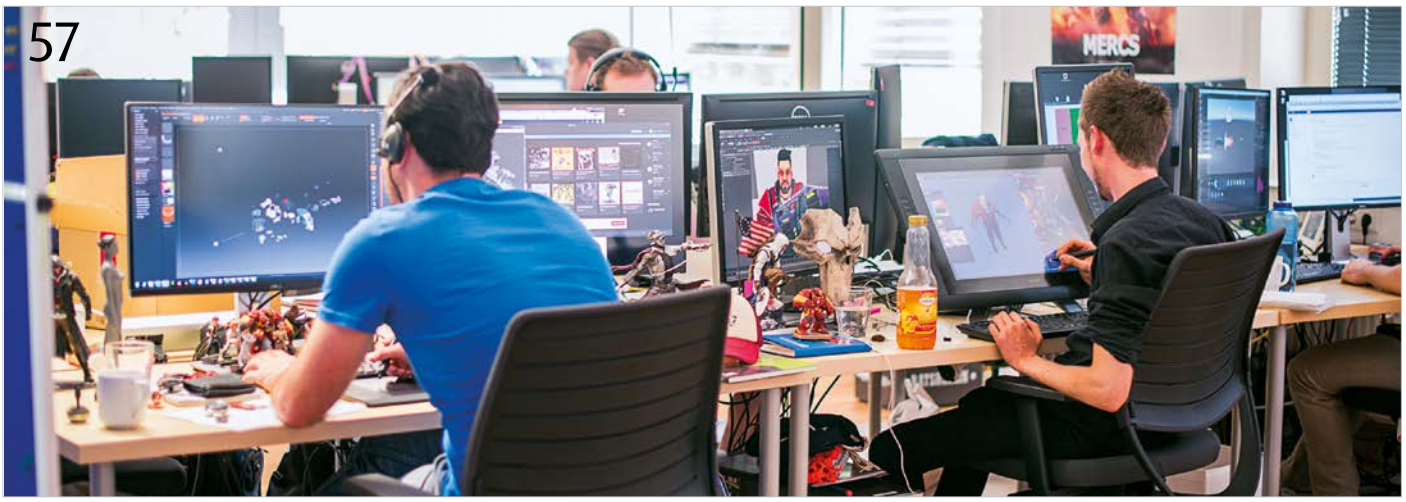
Spring Boot Starter
Komfortable Modularisierung und Konfiguration

4 191978 304903 04

iJUG Verbund

Anwendung im laufenden Betrieb verwalten

Business Process Management auf Open-Source-Basis



Ein Blick hinter die Kulissen eines Spiele-Unternehmens

Ansible – warum Konfigurationsmanagement auch für Entwickler interessant sein kann

Sandra Parsick

Das automatisierte Konfigurieren von Servern ist dank Orchestrierungswerkzeugen wie Puppet und Chef heute kein Problem mehr. Doch diese eignen sich wenig für die regelmäßige Verteilung von typischen Java-Webapplikationen. Ansible hat dieses Problem erkannt und liefert Lösungen für das Konfigurationsmanagement und die Software-Verteilung aus einer Hand.

Der Artikel zeigt am Beispiel einer Infrastruktur für eine Java-Webapplikation die Funktionsweise von Ansible. Zusätzlich wird darauf eingegangen, warum Ansible auch für Entwickler interessant sein kann. Dabei ist zu erkennen, wie Continuous Deployment auch in einer klassischen Unternehmensstruktur umsetzbar ist.

Ansible beschreibt sich selbst als ein Werkzeug für das Konfigurationsmanagement, für die Verteilung von Software und für die Ausführung von Ad-hoc-Kommandos. Der Begriff „Konfigurationsmanagement" kennt in der IT mehrere Aspekte. Eine Definition lautet: „Das Konfigurationsmanagement umfasst alle technischen, organisatorischen und beschlussfassenden Maßnahmen und Strukturen, die sich mit der Konfiguration (Spezifikation) eines Produkts befassen" *[1]*. In der IT kann sich das auf die Konfiguration der Software beziehen, etwa im Hinblick darauf, welche Datenbank die Anwendung wie benutzen soll. Außerdem kann damit die Konfiguration der Hardware gemeint sein („Wie viel CPU-Kerne soll der Server erhalten?") oder auch die Konfiguration eines Systems („Welche Software soll wie installiert sein?").

Bei Ansible geht es um die Konfiguration von Systemen. Die Idee ist, dass die Konfigurationen per Skript beschrieben sind („Infrastructure as Code"); mithilfe dieser Skripte konfiguriert Ansible automatisiert die jeweiligen Zielmaschinen *(siehe Abbildung 1)*.

Das Thema klingt bisher eher typisch für System-Administratoren als für Entwickler. Daher ein kurzer Einblick, wie die Zusammenarbeit zwischen System-Administratoren und Entwickler in einigen Unternehmen aussieht.

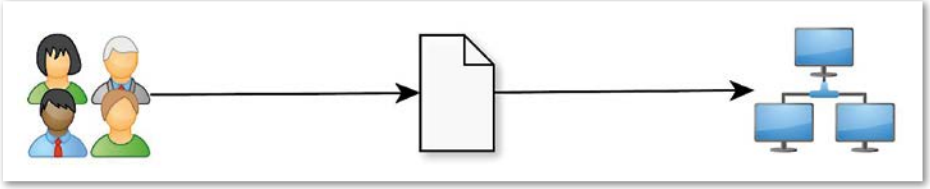


Abbildung 1: „Infrastructure as Code"

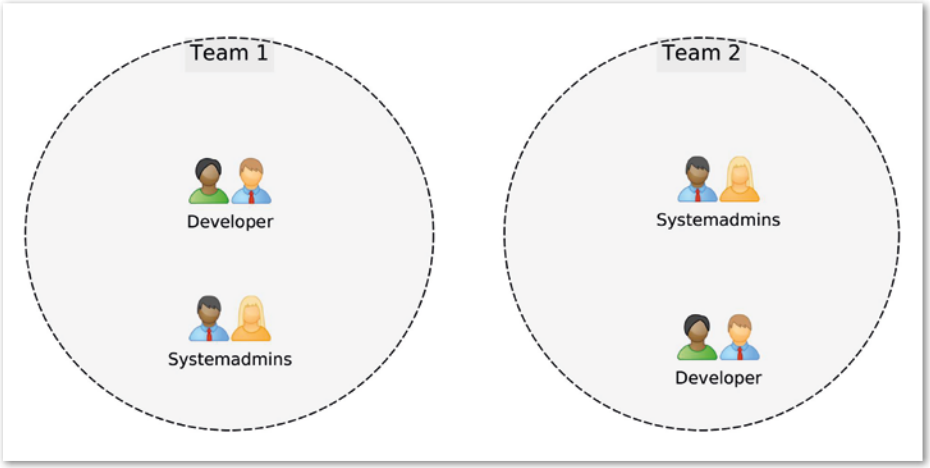


Abbildung 2: Cross-funktionale Teams

Motivation für Entwickler

Wenn man der Fachpresse Glauben schenken kann, wird in den meisten Unternehmen eine DevOps-Kultur gelebt. Es existieren kleine, cross-funktionale Teams, in denen Entwickler und System-Administratoren Hand in Hand zusammenarbeiten und in denen die Teams die Verantwortlichkeiten tragen – angefangen von der Entwicklung bis hin zum Betrieb der Software *(siehe Abbildung 2)*.

Das mag vor allem für Startup-Unternehmen zutreffen, doch in den meisten Unternehmen herrscht eher das klassische Bild; Entwicklung und Betrieb sind in zwei Abteilungen aufgeteilt, die unter Umständen unterschiedliche Ziele verfolgen *(siehe Abbildung 3)*.

Wie die Kommunikationswege zwischen den Abteilungen aussehen können, wird anhand einer Bestellung und Konfiguration eines neuen Testservers beschrieben, der auf Linux basiert *(siehe Abbildung 4)*. Meist erfolgt die Bestellung über ein Ticketsystem oder per E-Mail, in der beschrieben ist, welche Software (wie Oracle Java, Tomcat etc.) das Entwicklungsteam auf dem Server haben möchte und wie sie konfiguriert sein soll.

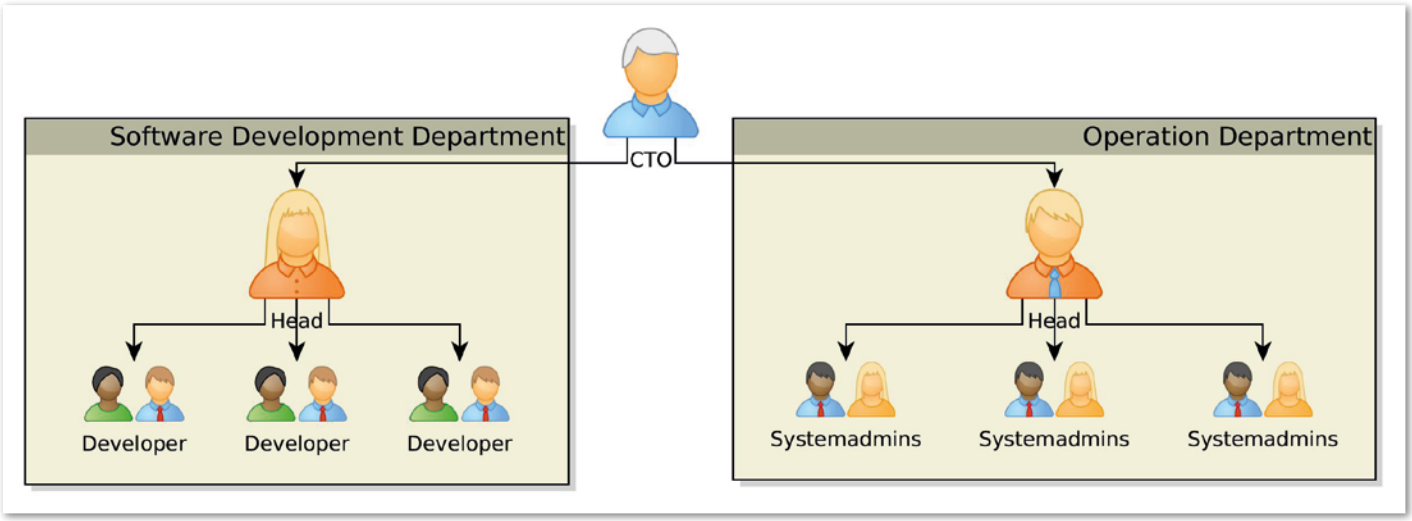


Abbildung 3: Klassische Trennung von Entwicklung und Betrieb

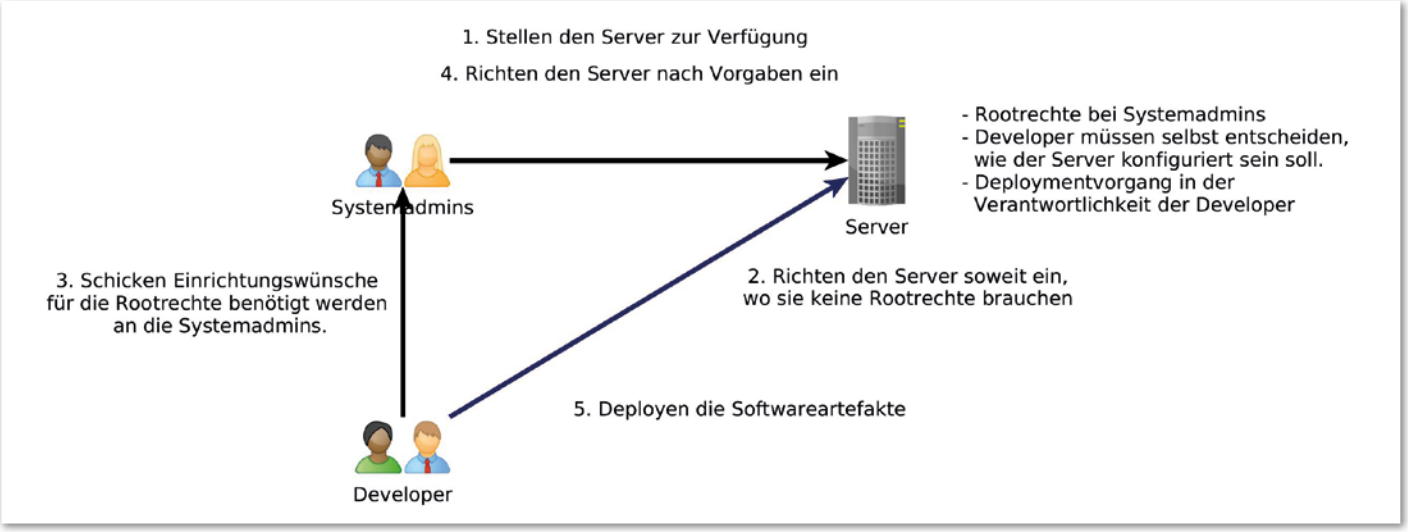


Abbildung 4: Prozess zwischen Entwicklung und Betrieb

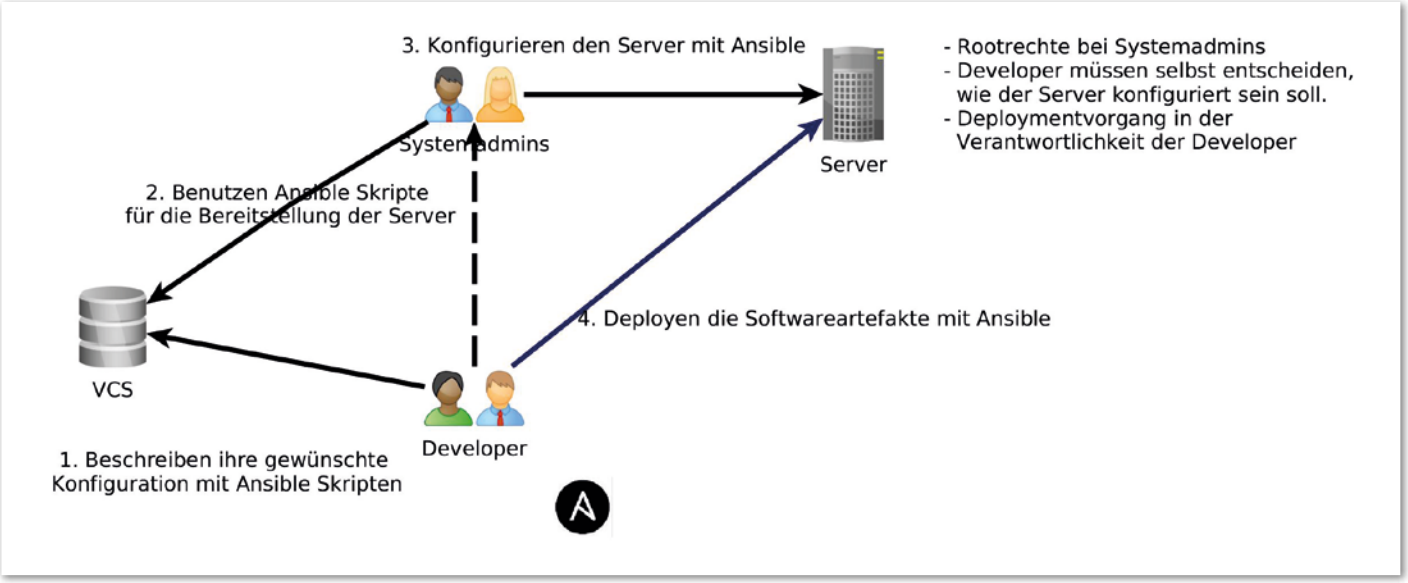


Abbildung 5: Lösungsidee mit Ansible

Oft werden nicht die aktuellsten Versionen installiert, da nur ältere Versionen in den Package-Repositories vorhanden sind. Hinzu kommt, dass die Distributionshersteller die Pakete „Linux-konform" bereitstellen, also aus Entwicklersicht verteilt auf das ganze System. Im besten Fall einigen sich die Betriebs- und die Entwicklungsabteilung darauf, dass die Entwickler die Java-Anwendungen selbst betreuen. Dann installieren und konfigurieren die Entwickler die Anwendungen manuell. Doch sobald der zweite oder auch dritte Test-Server angeschafft und bereitgestellt wird, kommt es zu Abweichungen, da der Mensch einfach nicht dazu geschaffen ist, monoton Sachen zu wiederholen.

Dasselbe geschieht bei der manuellen Verteilung der Anwendungen in Form von WAR-Dateien. Immer wieder hakt irgendetwas, weil eine Kleinigkeit vergessen wurde. Hier kann Ansible eine große Hilfe sein, da die Konfigurations- und Verteilungsschritte mithilfe verständlicher Skripte beschrieben werden können und für die Ausführung der Skripte derselbe Mechanismus gebraucht wird wie bei dem manuellen Vorgang, nämlich ein SSH-Zugang *(siehe Abbildung 5)*.

Die Funktionsweise

Ansible ist in Python geschrieben. Um es zu benutzen, ist Python nicht zwangsläufig erforderlich, da die Ansible-Skripte in YAML geschrieben sind. YAML ist eine Abstraktion von JSON mit dem Unterschied, dass sie besser lesbar ist *[2]*. Die Ansible-Skripte werden „Playbooks" genannt.

Damit Ansible eingesetzt werden kann, muss auf den Zielmaschinen Python 2.x installiert und ein Zugriff auf diese Maschinen über SSH möglich sein. Ansible ist nur auf der Maschine zu installieren, von der die Playbooks ausgeführt werden („Host Maschine"); sie muss ein Linux-System sein. Für Windows auf den Zielmaschinen bietet Ansible nur eine rudimentäre Unterstützung an. Da Python von System-Administratoren gerne für ihre eigenen Skripte benutzt wird, ist die Wahrscheinlichkeit recht hoch, dass Python standardmäßig auf den Linux-Servern installiert ist.

Wenn die in YAML geschriebenen Playbooks ausgeführt werden, übersetzt Ansible diese in Python-Skripte und führt sie auf den Zielmaschinen aus. Anschließend werden sie wieder von den Zielmaschinen entfernt.

Ansible wird in den nächsten Abschnitten anhand einer Infrastruktur für eine klassische Java-Web-Anwendung vorgestellt. Dabei werden ein Tomcat 8 und ein OpenJDK

installiert. Die Anwendung wird als WAR-Datei auf dem Tomcat eingesetzt.

Playbook für das Setup

Das in *Listing 1* gelistete Playbook „setup-app.yml" beschreibt die Konfiguration für die Installation von Tomcat 8 und OpenJDK. Jedes Playbook fängt mit der Beschreibung an, für welche Server („hosts") dieses Playbook gilt. Das kann eine Auflistung einzelner Hostnamen sein oder eine Bezeichnung für eine Gruppe von Servern. Diese Gruppe wird in einer eigenen Datei beschrieben, dem sogenannten „inventory". Dazu später mehr.

Optional lassen sich Variablen definieren („vars"), die später im Playbook vorkommen („{{ variabel_name }}"). Danach werden die Schritte beschrieben, wie der Server konfiguriert werden muss („tasks"). Ansible arbeitet diese Tasks von oben nach unten ab. Sobald ein Task fehlschlägt, ist die komplette Ausführung beendet – ohne die schon ausgeführten Tasks rückgängig zu machen.

Jeder Task beschreibt einen Aufruf eines sogenannten „module". Das sind fertige Skripte, die den Server konfigurieren. Sie müssen mindesten zwei Bedingungen erfüllen: Idempotent sein, ein mehrmaliges Ausführen muss also immer dasselbe Ergebnis liefern, und ihre Rückgabewerte müssen im JSON-Format sein. Die Skripte werden mithilfe von Key-Value-Paaren parametrisiert.

Die Skriptsprache der Module ist Ansible egal. Einzige Bedingung ist, dass ein Interpreter für die gewählte Sprache auf den Zielmaschinen installiert ist. Man wird aber sehr selten in die Lage kommen, eigene Module zu schreiben *[3]*, da Ansible von Haus aus eine beträchtliche Anzahl fertiger Module mitbringt *[4]*.

Im *Listing 1* wird als erster Task das Modul „apt" aufgerufen. Es soll über den Paket-Manager „apt" dafür sorgen, dass das Paket „openjdk-7-jdk" („name=openjdk-7-jdk") auf der Zielmaschine installiert ist („state=present"). Wenn also das angeforderte Paket schon installiert ist, dann macht das Modul nichts, ansonsten installiert es das Paket über den Paket-Manager.

Da für die Installation meist „sudo"-Rechte erforderlich sind, bekommt dieser Task zwei weitere Anweisungen („become: yes" und „become_method: sudo") mit, damit Ansible ihn mit „sudo" ausführt. Jeder Task kann mit einer Beschreibung versehen werden („name"). Sie vereinfacht es später bei der Ausführung festzustellen, welcher Task gerade ausgeführt wird.

Manchmal soll ein Task auf der Host-Maschine ausgeführt werden und nicht auf der Ziel-Maschine. Dafür gibt es „local_action", die als Eingabeparameter den Modulnamen und seine Parameter bekommt. Im *Listing 1* ist es der Task „Download Tomcat 8", der ein Tomcat-8-Archiv auf der Host-Maschine herunterlädt. Tomcat soll als Service laufen, dafür wird ein „init.d"-Skript benötigt.

Ansible bietet ein Modul „copy" an, das Dateien vom Host zur Ziel-Maschine kopiert (im *Listing 1* Task „install init.d script for tomcat"). Manchmal muss man Dateien beim Kopieren noch auf die jeweilige Ziel-Maschine anpassen, dafür gibt es das Modul „template". Dabei werden Dateien vom Host zur Ziel-Maschine kopiert und mithilfe einer Template-Engine entsprechend angepasst. Ansible benutzt dafür Jinja2 *[5]*, eine Template-Engine für Python.

In *Listing 1* soll anhand der Template-Datei „setenv.sh.j2" im Task „setup setenv.sh" mit „CATALINA_OPTS="{{ catalina_opts }}"" eine „setenv.sh"-Datei erzeugt werden. Die Besonderheit an diesem Task ist, dass er nur ausgeführt werden soll, wenn die Variable „catalina_opts" definiert ist („when: catalina_opts is defined").

Damit Tomcat auch gestartet und gestoppt werden kann, müssen alle Tomcat-Shell-Skripte ausführbar sein. Dies stellt das Modul „file" sicher (im *Listing 1* die Task „ensure tomcat scripts are executable"). Normalerweise muss dieser Task für jede Datei einzeln definiert sein. Ansible bietet aber eine Möglichkeit an, diesem Task eine Liste an Werten mitzugeben, für die ein Task wiederholt werden soll („with_items"). Im Beispiel wird diese Liste mithilfe eines vorherigen Task („shell: ls /opt/{{ tomcat_base_name }}/bin/*.sh") er-

```
- hosts: application-server
  vars:
    tomcat_version: 8.0.24
    tomcat_base_name: apache-tomcat-{{ tomcat_version }}
    #catalina_opts: "-Dkey=value"

  tasks:
    - name: install java
      apt: name=openjdk-7-jdk state=present
      become: yes
      become_method: sudo

    - name: Download Tomcat 8
      local_action: get_url url="http://archive.apache.org/dist/tomcat/
tomcat-8/v{{ tomcat_version }}/bin/{{ tomcat_base_name }}.tar.gz" dest=/tmp

    - name:
      file: name=/opt mode=777
      become: yes
      become_method: sudo

    - name: Install Tomcat 8
      unarchive: src=/tmp/{{ tomcat_base_name }}.tar.gz dest=/opt creates=/
opt/{{ tomcat_base_name }} owner=vagrant group=vagrant

    - name: Set link to tomcat 8
      file: src=/opt/{{ tomcat_base_name }} dest=/opt/tomcat  state=link
force=yes

    - name: setup setenv.sh
      template: dest="/opt/{{ tomcat_base_name }}/bin/setenv.sh"
src="roles/tomcat8/templates/setenv.sh.j2" mode=755
      when: catalina_opts is defined

    - shell: ls /opt/{{ tomcat_base_name }}/bin/*.sh
      register: tomcat_scripts
      ignore_errors: yes

    - name: ensure tomcat scripts are executable
      file: name={{item}} mode=755
      with_items: tomcat_scripts.stdout_lines

    - name: install init.d script for tomcat
      copy: src=roles/tomcat8/files/init.d/tomcat dest=/etc/init.d/tomcat
owner=vagrant group=vagrant mode=755
      become: yes
      become_method: sudo
```

Listing 1

mittelt und in der Variable „tomcat_scripts“ gespeichert („register: tomcat_scripts“), durch die dann im nächsten Task iteriert wird.

Um das Playbook auszuführen, muss in der Konsole „ansible-playbook setup-app.yml -u remote-user -i inventories/test“ aufgerufen werden. Der Parameter „-u“ definiert, mit welchem Benutzer das Playbook auf der Zielmaschine ausgeführt werden soll. Er ist gleichzeitig auch der SSH-Login-Benutzer. Der Parameter „-i“ definiert, welches Inventory für das Playbook benutzt wird.

Inventories

Wie im letzten Abschnitt beschrieben, fängt jedes Playbook mit einer Auflistung an, für welche Server dieses Playbook gelten soll. Die Auflistung kann auch Gruppen von Servern beinhalten. Diese werden in einem sogenannten „inventory“ genauer spezifiziert *(siehe Listing 2)*. Als Format wird INI benutzt *[6]*. Gruppen werden mit „[Gruppenname]“ definiert und dann folgt die Auflistung der Server (als IP-Adresse, Hostname etc.), die zu dieser Gruppe gehören. Zusätzlich lassen sich Gruppen von Gruppen bilden („[Gruppe_von_Gruppen_Name:children]“), die dann als Auflistung Gruppen haben. Zusätzlich zu den Servernamen können Variable als Key-Value-Paare definiert werden, die dann für den speziellen Server gelten. Sollen Variable für eine ganze Gruppe gelten, werden sie nach „[Gruppenname:vars]“ als eine Auflistung von Key-Value-Paaren definiert.

Mithilfe mehrerer Inventory-Dateien lassen sich Test- und Produktionsumgebung voneinander getrennt verwalten. Die Inventories haben beispielsweise dieselben Gruppen definiert, es werden dort aber andere Server aufgelistet. So können dieselben Playbooks für beide Umgebungen benutzt werden und der Parameter „-i“ beim Aufruf der Playbooks steuert, ob sie Richtung „Produktion“ oder „Test“ ausgeführt werden sollen.

Role

Aus Entwicklersicht lassen sich die Tasks für die Tomcat-Installation aus *Listing 1* gut zu einer Einheit gruppieren, da die Tasks allein betrachtet nicht viel Sinn ergeben. In Ansible können solche Einheiten mithilfe sogenannter „roles“ gebildet werden *[7]*. Dafür werden die Roles anhand einer festen Projektstruktur definiert *(siehe Listing 3)*. Dabei bildet jeder Unterordner von „roles“ eine Role ab; es werden zwei Roles definiert, „jdk“ und „tomcat8“.

```
[application-server]
192.168.33.10
ubuntu-server db_host=mysql01

[mysql-db-server]
mysql1
mysql2

[oracle-db-server]
db-a.oracle.company.com
db-b.oracle.company.com

[database-server:children]
mysql-db-server
oracle-db-server

[application-server:vars]
message="Welcome"

[database-server:vars]
message="Hello World!"
```

Listing 2

```
.
├── inventories
│   ├── production
│   └── test
├── roles
│   ├── jdk
│   │   └── tasks
│   │       └── main.yml
│   └── tomcat8
│       ├── defaults
│       │   └── main.yml
│       ├── files
│       │   └── init.d
│       │       └── tomcat
│       ├── tasks
│       │   └── main.yml
│       └── templates
│           └── setenv.sh.j2
└── setup-app.yml
```

Listing 3

```
- hosts: application-server
  roles:
    - jdk
    - { role: tomcat8, tomcat_ver-
sion: 8.0.30 }
```

Listing 4

```
- hosts: application-server
  roles:
    - {role: deploy-on-tomcat,
webapp_source_path: ./demo-app-
ansible-deploy-1.0-SNAPSHOT.war,
webapp_target_name: demo }
```

Listing 5

```
.
├── inventories
│   ├── production
│   └── test
├── roles
│   ├── deploy-on-tomcat
│   │   ├── defaults
│   │   │   └── main.yml
│   │   └── tasks
│   │       ├── cleanup-webapp.yml
│   │       ├── deploy-webapp.yml
│   │       ├── main.yml
│   │       ├── start-tomcat.yml
│   │       └── stop-tomcat.yml
└── deploy-demo.yml
```

Listing 6

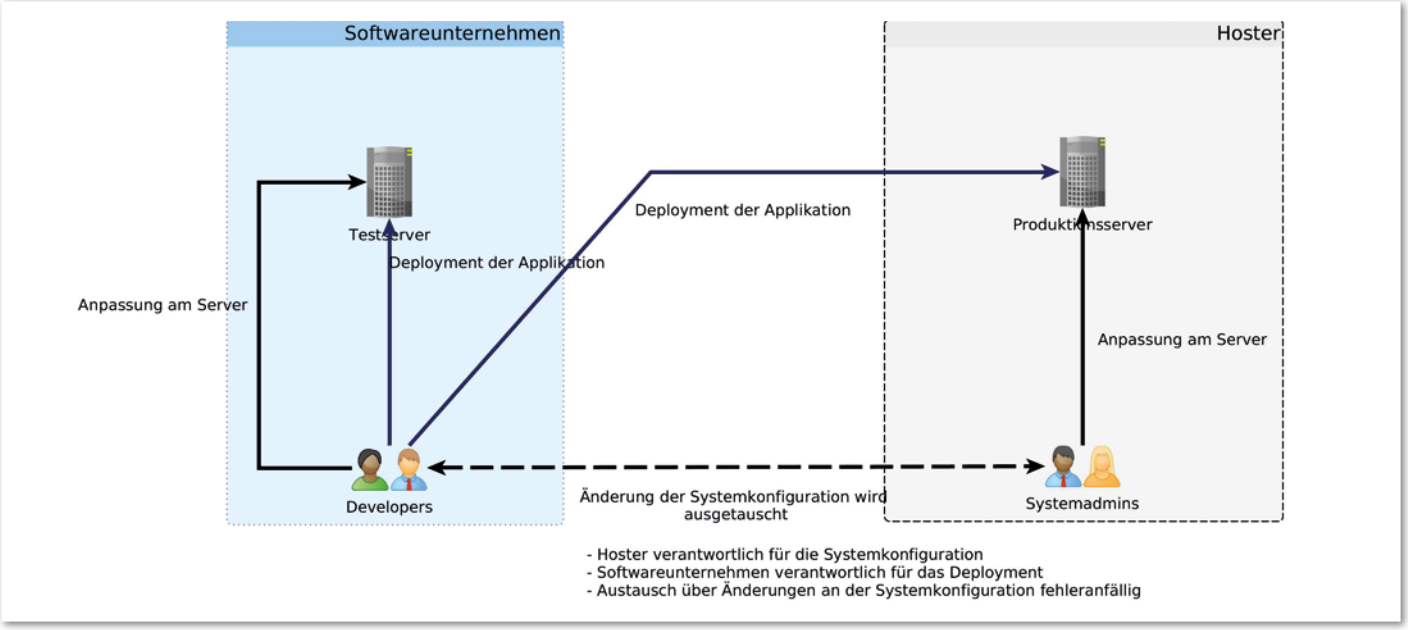


Abbildung 6: Prozess zwischen Entwicklung und externem Hoster

Alle Tasks einer Role sind in „tasks/main.yml" definiert. Dateien, die unverändert auf die Zielmaschine kopiert werden sollen, werden unter „files" abgelegt; Template-Dateien unter „template". In „default/main.yml" sind Defaultwerte für Variablen definiert. Das Playbook aus *Listing 1* kann dann, wie in *Listing 4* gezeigt, vereinfacht werden.

Deployment der Webapplikation

Nachdem Tomcat und Java installiert sind, wird jetzt die Webapplikation in Form einer WAR-Datei eingerichtet. Dazu definiert man ein separates Playbook „deploy-demo.yml" *(siehe Listing 5)*. Es ruft die Role „deploy-on-tomcat" auf *(siehe Listing 6)*. Deren „main.yml"-Datei definiert die Tasks *(siehe Listing 7)* für das Deployment. Hier sind die Tasks noch weiter in einzelne Dateien gruppiert, die durch die „include"-Anweisungen zusammengeführt werden. Dann sieht der Ablauf für das Deployment folgendermaßen aus:

1. Tomcat herunterfahren („stop-tomcat.yml", *siehe Listing 8)*
2. Alte Webapplikation löschen („cleanup-webapp.yml", *siehe Listing 9)*
3. Neue Webapplikation hochladen („deploy-webapp.yml", *siehe Listing 10)*
4. Tomcat hochfahren („start-tomcat.yml", *siehe Listing 11)*

Auch dieses Playbook wird mit „ansible-playbook" ausgeführt („ansible-playbook deploy-demo.yml -u remote-user -i inventories/test").

```
- include: stop-tomcat.yml
- include: cleanup-webapp.yml
- include: deploy-webapp.yml
- include: start-tomcat.yml
```

Listing 7

```
- name: stop tomcat
  command: service tomcat stop

- name: wait tomcat shutdown
  wait_for: port=8080 state=stopped timeout=60
```

Listing 8

```
- name: cleanup {{ webapp_target_name }}
  file: name={{tomcat_app_base}}/{{ webapp_target_name }} state=absent
```

Listing 9

```
- name: delete previous backup
  file: path={{ tomcat_app_base }}/{{ webapp_target_name }}.war.previous
state=absent

- name: create new backup
  command: mv {{ tomcat_app_base }}/{{ webapp_target_name }}.war {{ tom-
cat_app_base }}/{{ webapp_target_name }}.war.previous
  ignore_errors: yes

- name: copy webapp {{ webapp_source_path }} to {{ webapp_target_name }}
  copy: src={{ webapp_source_path }} dest={{ tomcat_app_base }}/{{ webapp_
target_name }}.war mode=660
```

Listing 10

```
- name: start tomcat
  command: service tomcat start

- name: wait for tomcat to start
  wait_for: port=8080 timeout=60
```

Listing 11

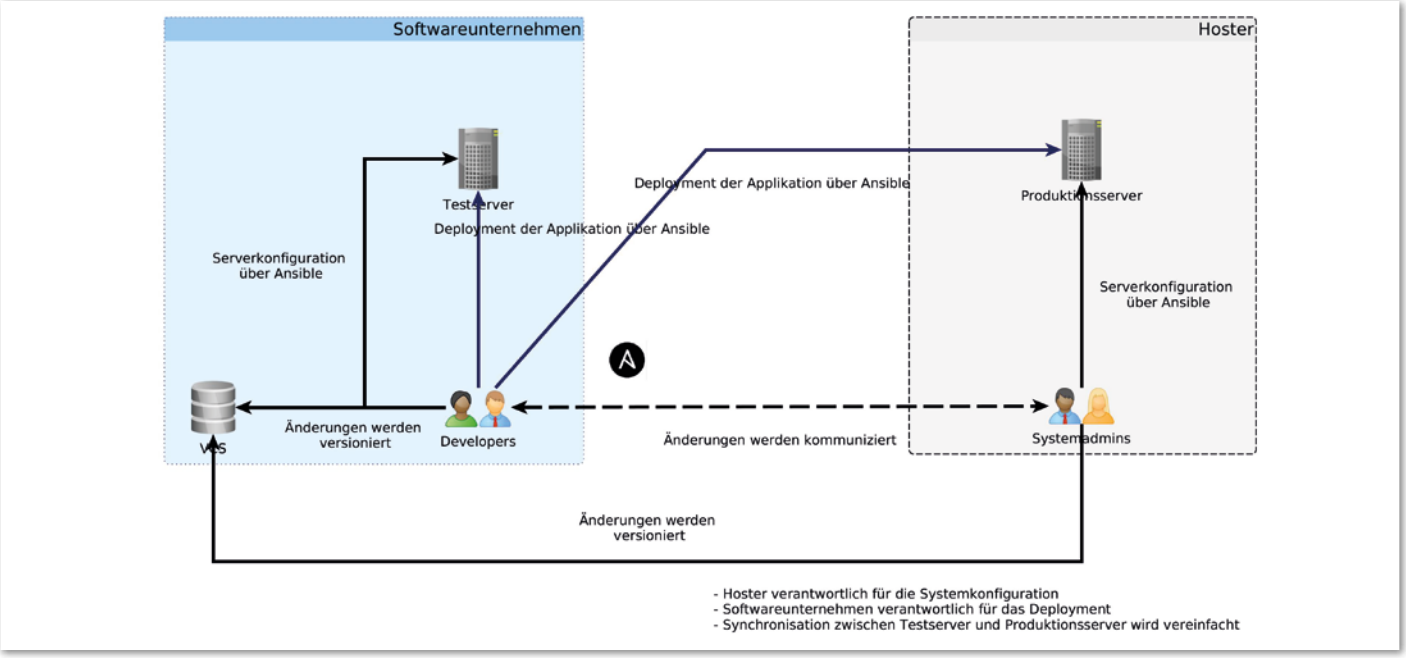


Abbildung 7: Lösungsidee für die Kooperation mit Hoster

Weitere Einsatzszenarien aus Entwicklersicht

Die in der Motivation vorgestellte Ausgangssituation *(siehe Abbildung 4)* kann auch in einer leicht abgewandelten Variante existieren. In dieser Variante sind die Kommunikationswege zwischen Entwicklung und Betriebsabteilung genauso wie in der vorgestellten Ausgangssituation. Der Unterschied liegt darin, dass die Betriebsabteilung ein Konfigurationsmanagement-Werkzeug wie zum Beispiel Puppet im Einsatz hat. Dennoch gibt es dieselben Probleme. Eine Lösung wäre, dass die Entwicklung Zugang zu den Puppet-Skripten bekommt und ihre Änderungswünsche selbst implementiert. Die Betriebsabteilung kontrolliert die Änderungen und führt sie aus. Dieses Vorgehen beschleunigt die Zeit zwischen Anforderung und Umsetzung. Die Automatisierung des Deployments kann wiederum in Ansible erfolgen.

Eine weitere Ausgangssituation kann sein, dass die Produktionsserver bei einem externen Hoster betrieben werden, aber die Testserver im eigenen Unternehmen *(siehe Abbildung 6)*.

Damit die Tests in einer produktionsnahen Umgebung ausgeführt werden können, ist die Konfiguration der Server zwischen Hoster und eigenem Unternehmen zu kommunizieren. Meist erfolgt dies per E-Mail und/oder über ein Ticketsystem. In diesem Vorgehen sind jedoch einige Fallstricke versteckt. Beide Seiten müssen immer daran denken, die andere Seite zu informieren, wenn Änderungen an der Konfiguration erfolgen. Das wird gerne im Eifer des Gefechts vergessen. Dann kann die Umsetzung sich unterscheiden.

Je nachdem, wer das Deployment auf den Produktionsserver machen darf, kann es auch hier Unterschiede in der Umsetzung geben. Auch wenn die Verantwortlichkeit des Deployments bei den Entwicklern liegt, wird auch hier meist nur ein SSH-Login bereitgestellt und das Deployment erfolgt manuell. Das alles zusammengenommen führt oft dazu, dass mit der Zeit die Umsetzung der Konfiguration und des Deployments auf dem Test- und auf dem Produktionsserver auseinanderlaufen.

Eine Lösung wäre, dass der Hoster und die eigene Entwicklung Zugriffe auf gemeinsame Ansible-Playbooks haben, die die Konfiguration der Server sowie das Deployment für beide Umgebungen beschreiben. Mit deren Hilfe ist sichergestellt, dass die Test- und die Produktionsserver gleich konfiguriert sind und dass das Deployment in der Produktion genauso abläuft wie in den Tests *(siehe Abbildung 7)*.

Fazit

Dieser Artikel gibt einen kleinen Überblick über Ansible und zeigt auf, dass es auch für Entwickler lohnt, sich mit Konfigurationsmanagement-Werkzeugen auseinanderzusetzen. Wer tiefer in Ansible einsteigen möchte, dem sei die sehr gute Dokumentation von Ansible *[8]* und *[9]* empfohlen. Alle hier vorgestellten Playbooks findet man auch auf GitHub *[10]*.

Quellen

[1] *https://www.projektmagazin.de/glossarterm/konfigurationsmanagement*
[2] *https://en.wikipedia.org/wiki/YAML*
[3] *http://docs.ansible.com/ansible/developing_modules.html*
[4] *http://docs.ansible.com/ansible/list_of_all_modules.html*
[5] *http://jinja.pocoo.org/docs/dev*
[6] *https://en.wikipedia.org/wiki/INI_file*
[7] *http://docs.ansible.com/ansible/playbooks_roles.html#roles*
[8] *http://docs.ansible.com*
[9] *Lorin Hochstein: Ansible: Up and Running. O'Reilly Media (2014)*
[10] *https://github.com/sparsick/ansible-java-aktuell*

Sandra Parsick
info@sandra-parsick.de

Sandra Parsick, geb. Kosmalla, ist als freiberufliche Software-Entwicklerin und Consultant im Java-Umfeld tätig. Seit dem Jahr 2008 beschäftigt sie sich mit agiler Software-Entwicklung in verschiedenen Rollen. Ihre Schwerpunkte liegen im Bereich der Enterprise-Anwendungen, agilen Methoden, Software Craftsmanship und in der Automatisierung von Softwareentwicklungs-Prozessen. In ihrer Freizeit engagiert sie sich in der Softwerkskammer Dortmund.