Ansible-Kochbuch

Rezepte für die Verwendung von Ansible

Jörg Kastning

17. Dezember 2016

Inhaltsverzeichnis

1 Einleitung

Dieser Text bietet eine druckbare Fassung meiner Ansible-Artikel im PDF-Format. Alle hierin veröffentlichten Artikel sind zuerst auf meiner Webseite [1] erschienen.

2 IT-Automation für Jedermann

Ansible [10] [16] ist eine Open-Source-Plattform zur Orchestrierung und allgemeinen Konfiguration und Administration von Computern. Ansible soll dabei helfen, Konfigurationsprozesse zu automatisieren und die Administration multipler Systeme zu vereinfachen. Damit verfolgt Ansible im Wesentlichen die gleichen Ziele wie z.B. Puppet [18], Chef [17] und Salt [19].

Ansible hat gegenüber den genannten Systemen das Alleinstellungsmerkmal, dass auf den verwalteten Rechnern kein Agent installiert werden muss. Der Zugriff erfolgt ausschließlich über das SSH-Protokoll. Dies ist ein Vorteil, da bei der Einführung von Ansible SSH-Key-Authentifizierungsverfahren genutzt werden können, die häufig schon existieren. Ein weiterer Vorteil ergibt sich in dem Fall, wenn man Ansible nicht mehr nutzen möchte. Es bleiben keine Softwarekomponenten auf den verwalteten Rechnern zurück, die wieder deinstalliert werden müssen.

Mir gefällt auf den ersten Blick, dass mir die Lösung zwei Wege bietet, um meine Systeme zu konfigurieren. Zum einen kann ich Playbooks [8] nutzen, welche gewissermaßen das Pendant der Puppet Manifeste darstellen. Zum anderen existiert ein Ad-hoc-Modus, welcher es erlaubt, die Konfiguration von Zielsystemen bei Bedarf anzupassen, ohne zuvor ein Rezept bzw. Playbook erstellt zu haben.

Die Playbooks selbst werden in YAML-Syntax [20] verfasst. Dadurch wird eine gute Lesbarkeit sichergestellt. Zudem ist die Syntax recht einfach gehalten und daher schnell zu erlernen.

Für häufige Anwendungsfälle wie z.B. das Anlegen eines Benutzers, die Verteilung von SSH-Keys, die Installation verfügbarer Updates/Patches und noch vieles mehr, bringt Ansible vorgefertigte Module [15] mit. Diese können direkt auf den Zielsystemen oder durch die Verwendung von Playbooks ausgeführt werden.

Die oben genannten Aspekte motivieren mich, zu evaluieren, ob Ansible für den Einsatz in meinem beruflichen Umfeld geeignet ist. In diesem Blog werden in unregelmäßigen Abständen weitere Artikel erscheinen, in denen ich Erkenntnisse aus meiner Evaluierung festhalte und einige kurze Beispiele zur Nutzung konkreter Module wiedergebe.

Wer dieser Reihe folgt und sich gern mit seinen Erfahrungen einbringen möchte, darf dies gerne tun. Ich freue mich jederzeit über Kommentare, Ideen und Anregungen.

3 Linux-Benutzerkonten mit Ansible verwalten

Wie bereits in [1] erwähnt, beschäftige ich mich aktuell mit der Evaluierung von Ansible.

In diesem Artikel dokumentiere ich ein Beispiel, wie das Modul user zur Administration von Linux-Benutzerkonten verwendet werden kann. Der Link zur offiziellen Dokumentation des user-Moduls befindet sich bei [7].

Hinweis: Die folgende Konfiguration ist nicht zur Nachahmung empfohlen. Es handelt sich dabei um meine ersten Schritte mit Ansible. Es ist daher wahrscheinlich, dass die Konfiguration noch Fehler enthält, nicht optimiert ist und ein großes Verbesserungspotenzial besitzt. Die Nachahmung erfolgt ausdrücklich auf eigene Gefahr. Über Hinweise, wie man es besser machen kann, freue ich mich jederzeit.

3.1 Anforderungen

- Es soll ein lokales Benutzerkonto angelegt werden
- Der Benutzer soll Mitglied der Gruppe "wheel"werden
- Es soll ein initiales Passwort für das Benutzerkonto gesetzt werden
- Ein öffentlicher SSH-Schlüssel soll in die Datei authorized_keys des zu erstellenden Benutzers eingetragen werden
- Konfiguration des SSH-Daemons

3.2 Vorbereitung

Abgeleitet aus den „Best Practices„ [4] verwende ich auf meiner Spielwiese folgende Verzeichnisstruktur:

Listing 3.1: Verzeichnisstruktur

```
1  ansible/
2  |-- autoupdate.txt            # ignore
3  |-- autoupdate.yml            # ignore
4  |-- check_reboot.             # playbook which sets the repos
5  |-- group_vars
6  |   `-- e-stage               # variables for group e-stage
7  |-- hosts                     # inventory file
8  |-- host_vars                 # for system specific variables
9  |-- roles
10 |   |-- common                # this hierarchy represents a "role"
11 |   |   |-- defaults
```

```
12 |   |   |-- files
13 |   |   |-- handlers
14 |   |   |   `-- main.yml       # handlers file
15 |   |   |-- meta
16 |   |   |-- tasks
17 |   |   |   `-- main.yml       # tasks file can include smaller files if
       warranted
18 |   |   `-- vars
19 |   `-- e-stage
20 |       |-- defaults
21 |       |-- files
22 |       |-- handlers
23 |       |-- meta
24 |       |-- templates
25 |       `-- vars
26 |-- create_johnd.yml           # playbook which creates user johnd
27 |-- site.yml                   # master playbook file
28 `-- staging                    # inventory file for staging environment
29 \begin{verbatim}
30 ansible/
31 |-- autoupdate.txt            # ignore
32 |-- autoupdate.yml            # ignore
33 |-- check_reboot.             # playbook which sets the repos
34 |-- group_vars
35 |   `-- e-stage               # variables for group e-stage
36 |-- hosts                     # inventory file
37 |-- host_vars                 # for system specific variables
38 |-- roles
39 |   |-- common                # this hierarchy represents a "role"
40 |   |   |-- defaults
41 |   |   |-- files
42 |   |   |-- handlers
43 |   |   |   `-- main.yml      # handlers file
44 |   |   |-- meta
45 |   |   |-- tasks
46 |   |   |   `-- main.yml      # tasks file can include smaller files if
       warranted
47 |   |   `-- vars
48 |   `-- e-stage
49 |       |-- defaults
50 |       |-- files
51 |       |-- handlers
52 |       |-- meta
53 |       |-- templates
54 |       `-- vars
55 |-- create_johnd.yml          # playbook which creates user johnd
56 |-- site.yml                  # master playbook file
57 `-- staging                   # inventory file for staging environment
58
59 17 directories, 11 files
```

Um das Passwort für den neuen Benutzer erstellen zu können, muss zuerst der passende Passwort-Hash generiert werden [11]:

Listing 3.2: Generierung eines Passowort-Hashs

```
1 python -c "from␣passlib.hash␣import␣sha512_crypt;␣import␣getpass;␣print␣
      sha512_crypt.encrypt(getpass.getpass())"
```

Der generierte Hash wird auf der Standardausgabe ausgegeben. Dieser wird für das Playbook benötigt.

3.3 Playbook und Tasks

Das Playbook ist sehr kurz gehalten und gibt im Wesentlichen nur an, welche Hosts auf welche Rolle gemapped werden sollen.

Listing 3.3: Ansible Playbook

```
# create_johnd.yml
---
- hosts: all
  roles:
    - common
```

Bei Verwendung dieses Playbooks werden die Tasks aus der Datei roles/common/tasks/main.yml ausgeführt, welche wie folgt aufgebaut wurde:

Listing 3.4: Datei: roles/common/tasks/main.yml

```
---
# Configure sshd_config on target system
- name: Enable Public-Key-Authentication
  lineinfile:
    dest: /etc/ssh/sshd_config
    regexp: "^PubkeyAuthentication"
    line: "PubkeyAuthentication␣yes"
    state: present
  notify:
    - reload sshd

- name: Set AuthorizedKeyFile
  lineinfile:
    dest: /etc/ssh/sshd_config
    regexp: "^AuthorizedKeysFile"
    line: "AuthorizedKeysFile␣␣␣␣␣␣.ssh/authorized_keys"
    state: present
  notify:
    - reload sshd

- name: Disable PasswordAuthentication
  lineinfile:
    dest: /etc/ssh/sshd_config
    regexp: "^PasswordAuthentication"
    line: "PasswordAuthentication␣no"
    state: present
  notify:
    - reload sshd

# Add user johnd with specific uid in group 'wheel'
- user: name=johnd comment="John␣Doe" uid=100 groups=wheel password="
    PASSWORDHASH" shell=/bin/bash append=yes state=present
  notify:
    - deploy ssh-pub-keys
```

Die in 3.4 spezifizierten Tasks setzen die folgenden Aufgaben um:

- Aktivierung der Public-Key-Authentication
- Spezifikation des Standard-AuthorizedKeysFile
- Deaktivierung der Password-Authentication

Was in 3.4 noch nicht zu sehen ist, ist die Verteilung eines öffentlichen SSH-Schlüssels. Dies wird von dem Handler deploy ssh-pub-keys übernommen.

Der Handler ist in der Datei roles/common/handlers/main.yml definiert. Es handelt sich dabei um einen Task, welcher nur ausgeführt wird, wenn der Benutzer erfolgreich erstellt wurde.

3.4 Fazit

Soweit zu meinen ersten Gehversuchen mit Ansible Playbooks. Meine Anforderungen können mit dem hier dokumentierten Playbook erfüllt werden.

In dem hier beschriebenen Beispiel habe ich angenommen, dass es sich bei dem zu erstellenden Benutzer um das erste lokale Benutzerkonto neben root handelt. Dieses soll über die Berechtigung verfügen, Kommandos mit sudo auszuführen und primär für die Arbeit auf dem System verwendet werden.

Die Verteilung des SSH-Schlüssels über einen Handler anzustoßen erscheint mir jedoch nicht optimal. Der Handler wird nur dann ausgeführt, wenn durch das user-Modul Änderungen am Zielsystem vorgenommen wurden. Weitere SSH-Schlüssel bzw. Änderungen an bereits vorhandenen SSH-Schlüsseln lassen sich auf diesem Weg nicht vornehmen.

Daher erscheint es mir sinnvoll, die Verteilung von SSH-Schlüsseln über das Modul authorized_key [3] in ein eigenes Playbook auszulagern.

4 Ansible - Das Modul yum_repository

Das Ansible Modul yum_repository [14] ist dazu geeignet, Repositories zu RPM-basierten Linux-Distributionen hinzuzufügen bzw. zu entfernen.

Auf meiner Spielwiese verwende ich folgende Konfiguration, um zwei Repositories auf den Servern der Gruppe ë-stage"hinzuzufügen.

Hinweis: Die folgende Konfiguration ist nicht zur Nachahmung empfohlen. Es handelt sich dabei um meine ersten Schritte mit Ansible. Es ist daher wahrscheinlich, dass die Konfiguration noch Fehler enthält, nicht optimiert ist und ein großes Verbesserungspotenzial besitzt. Die Nachahmung erfolgt ausdrücklich auf eigene Gefahr. Über Hinweise, wie man es besser machen kann, freue ich mich jederzeit.

4.1 Verzeichnisstruktur

Abgeleitet aus den "Best Practices"[4] verwende ich auf meiner Spielwiese folgende Verzeichnisstruktur:

Listing 4.1: Genutzte Verzeichnisstruktur

```
ansible/
|-- autoupdate.txt            # ignore
|-- autoupdate.yml            # ignore
|-- check_reboot.             # playbook which sets the repos
|-- group_vars
|   `-- e-stage               # variables for group e-stage
|-- hosts                     # inventory file
|-- host_vars                 # for system specific variables
|-- roles
|   |-- common                # this hierarchy represents a "role"
|   |   |-- defaults
|   |   |-- files
|   |   |-- handlers
|   |   |   `-- main.yml      #  handlers file
|   |   |-- meta
|   |   |-- tasks
|   |   |   `-- main.yml      #  tasks file can include smaller files if
    warranted
|   |   `-- vars
|   `-- e-stage
|       |-- defaults
|       |-- files
|       |-- handlers
|       |-- meta
|       |-- templates
|       `-- vars
|-- set_repos.yml             # playbook which sets the repos
|-- site.yml                  # master playbook file
`-- staging                   # inventory file for staging environment

17 directories, 11 files
```

4.2 Das Playbook

Das Playbook ist sehr kurz gehalten und gibt im Wesentlichen nur an, welche Hosts auf welche Rolle gemapped werden sollen.

Listing 4.2: Ansible-Playbook: set_repos.yml

```
# set_repos.yml
---
- hosts: e-stage
  roles:
    - e-stage
```

Das obige Playbook definiert, dass die Tasks aus der Datei roles/e-stage/tasks/main.yml ausgeführt werden sollen, welche wie folgt aufgebaut ist:

Listing 4.3: Tasks zum Hinzufügen von Repositories

```
---
- name: Add rhel-e-stage-repository
  yum_repository:
    name: rhel-e-stage
    description: Local RHEL-Repo for T-, E- and I-Stage
    baseurl: http://repo.example.com/rhel-e-stage/
    gpgcheck: yes
    gpgkey: file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release

- name: Add own-e-stage-repository
  yum_repository:
    name: own-e-stage
    description: Local Repo for T-, E- and I-Stage
    baseurl: http://repo.example.com/own-e-stage/
    gpgcheck: no
```

4.3 GroupVars

Nun existieren im Staging-Bereich häufig noch weitere Stages. So gibt es oftmals eine T-, E- und I-Stage. Dabei ist jeder Stage ein eigenes Stage-Repository zugeordnet. Das obere Playbook soll nun so erweitert werden, dass die Systeme in den unterschiedlichen Stages die dazugehörigen Stage-Repositories konfiguriert bekommen.

Zur Lösung greife ich dazu auf "Group Variablesßurück. Diese werden für die Server der Gruppe e-stage in der Datei group_vars/e-stage definiert:

Listing 4.4: Datei: group_vars/e-stage

```
# file: group_vars/e-stage
repo_name1: rhel-e-stage
repo_description1: Local RHEL-Repo for E-Stage
repo_baseurl1: http://repo.example.com/rhel-e-stage/

repo_name2: own-e-stage
repo_description2: Local OWN-Repo for E-Stage
repo_baseurl2: http://repo.example.com/own-e-stage/
```

Die Datei roles/e-stage/tasks/main.yml wird nun wie folgt angepasst:

Listing 4.5: Datei: roles/e-stage/tasks/main.yml

```
# Configure repositories on target hosts
- name: Add rhel-repository
  yum_repository:
    name: "{{ repo_name1 }}"
```

```
5       description: "{{ repo_description1 }}"
6       baseurl: "{{ repo_baseurl1 }}"
7       gpgcheck: yes
8       gpgkey: file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
9
10    - name: Add hrz-repository
11      yum_repository:
12        name: "{{ repo_name2 }}"
13        description: "{{ repo_description2 }}"
14        baseurl: "{{ repo_baseurl2 }}"
15        gpgcheck: no
```

Hier werden nun statt der spezifischen Werte für eine Stage die Group-Variablen verwendet. Nach diesem Beispiel können nun weitere Variablen unter group_vars für andere Stages erstellt werden. Den Hosts in den entsprechenden Hostgruppen, werden auf diesem Wege die passenden Repositories hinzugefügt.

5 Die Module copy und cron

In diesem Beitrag zu meiner kleinen Ansible-Reihe beschäftige ich mich mit den Modulen copy [5] und cron [6].

Hinweis: Die folgende Konfiguration ist nicht zur Nachahmung empfohlen. Es handelt sich dabei um meine ersten Schritte mit Ansible. Es ist daher wahrscheinlich, dass die Konfiguration noch Fehler enthält, nicht optimiert ist und ein großes Verbesserungspotenzial besitzt. Die Nachahmung erfolgt ausdrücklich auf eigene Gefahr. Über Hinweise, wie man es besser machen kann, freue ich mich jederzeit.

5.1 Anforderungen

In diesem Szenario sollen Shell-Skripte auf das Zielsystem kopiert werden. Anschließend sind Einträge in der crontab zu erstellen, um diese Shell-Skripte auszuführen. Beim Zielsystem handelt es sich um die Gruppe [i-stage], in der sich aktuell nur ein Host namens host-i1.example.com befindet.

5.2 Vorbereitungen

Die zu verteilenden Skripte werden auf der Ansible-Control-Machine im Verzeichnis /root/src/ gesammelt. Anschließend wurde eine Rolle für dieses Vorhaben erstellt. Die auf meiner Spielwiese verwendete Verzeichnisstruktur wurde bereits in den vergangenen Beiträgen [2] zu dieser Reihe beschrieben.

5.3 Umsetzung

Zur Umsetzung der Anforderungen wird ein Playbook erstellt, welches die geforderten Tasks ausführt. Der folgende Code-Block listet die benötigten Dateien inkl. ihres kommentierten Inhalts auf:

Listing 5.1: Zur Umsetzung benötigte Dateien

```
1  # Begin of file: local_scripts_i-stage.yml
2  ---
3  - hosts: i-stage
4    roles:
5      - local_scripts
6  # End of file ############################
7
8  # Begin of file: roles/local_scripts/tasks/main.yml
9  ---
10 ## Copy local script files to target hosts
```

```
11 - copy: src=/root/src/sript1.sh dest=/root/src/sript1.sh owner=root group=root
       mode=755
12 - copy: src=/root/src/sript2.sh dest=/root/src/sript2.sh owner=root group=root
       mode=755
13
14 ## Create cron jobs for local script files
15 - cron: name="local␣script1" minute="10" hour="1" job="/root/src/sript1.sh"
16 - cron: name="local␣script2" minute="12" hour="1" job="/root/src/sript2.sh"
17 # End of file ##############################
```

Nach der Ausführung des Playbooks liegen die Skripte auf dem oder den Zielsystemen im gewünschten Verzeichnis /root/bin/. Lässt man sich die Crontab des Benutzers root anzeigen, so erkennt man die von Ansible hinzugefügten Einträge:

Listing 5.2: Crontab des Benutzers root

```
1 # crontab -l
2 # DO NOT EDIT THIS FILE - edit the master and reinstall.
3 # (/tmp/crontabaf5r8i installed on Mon Jul 18 11:52:09 2016)
4 # (Cronie version 4.2)
5 #Ansible: local script1
6 10 1 * * * /root/bin/script1.sh
7 #Ansible: local script2
8 12 1 * * * /root/bin/script2.sh
```

5.4 Fazit

Wieder konnte eine Anforderung schnell und einfach mit Ansible umgesetzt werden. Ob dies bereits der Königsweg ist bzw. ob dieser überhaupt existiert, steht jedoch noch nicht fest. Anstatt die Dateien zuerst auf das Ziel zu kopieren und dort Cron-Einträge für die Ausführung zu erstellen, könnte auch das Modul script [9] genutzt werden. Dieses Modul transferiert ein lokales Skript zum Zielsystem und führt es dort aus.

Bisher habe ich mich gegen den Einsatz des script-Moduls entschieden, da mir noch nicht klar ist, ob das Skript auf dem Zielsystem verbleibt oder bei jeder Ausführung erneut übertragen wird. Letzteres wäre bei einer großen Anzahl Zielsysteme nicht unbedingt wünschenswert.

6 Patchmanagement für RHEL-Systeme

Als Linux-Distribution für den Betrieb im Rechenzentrum fiel unsere Wahl vor einiger Zeit auf Red Hat. Zu meinen Aufgaben gehört es, ein Patch-Management für diese Systeme zu entwickeln, welches die folgenden Anforderungen erfüllt:

1. An definierten Stichtagen sollen automatisiert fehlende Sicherheitsaktualisierungen auf den Systemen in den verschiedenen Stages eingespielt werden können.
2. Dabei sollen ausschließlich Pakete aktualisiert werden, für die Red Hat Security Advisory veröffentlicht wurden.
3. Wenn Pakete aktualisiert wurden, soll der Host anschließend neugestartet werden.

Ich habe mich entschieden, diese Anforderungen durch die Verwendung von Ansible zu erfüllen. Mit den Anregungen meines Arbeitskollegen habe ich dazu heute das folgende Playbook erstellt, welches die Rolle patch_rhel auf allen Hosts mit einem Red Hat Betriebssystem ausführt:

Listing 6.1: Playbook zum Filtern der RHEL-Systeme

```
1  ---
2  - hosts: all
3
4    tasks:
5      - name: Group by OS
6        group_by: key=os_{{ ansible_distribution }}
7        changed_when: False
8
9  - hosts: os_RedHat
10   roles:
11     - patch_rhel
```

Die Rolle patch_rhel besteht aus den beiden folgenden Dateien:

- roles/patch_rhel/tasks/main.yml
- roles/patch_rhel/vars/main.yml

Listing 6.2: Datei: roles/patch_rhel/vars/main.yml

```
1  ---
2  rhsa_to_install: RHSA-2016:1626,RHSA-2016:1628,RHSA-2016:1633
```

Obiges Listing gibt ein kurzes Beispiel, wie die Red Hat Security Advisory (RHSA) Nummern einer Variablen zugewiesen werden können. Die Variable rhsa_to_install wird dann in der Task-Datei verwendet.

Listing 6.3: Datei: roles/patch_rhel/tasks/main.yml

```
---
  - name: Install Red Hat Security Advisory (RHSA)
    command: yum -y update-minimal --advisory {{ rhsa_to_install }}
    register: yum_output
  - debug: var=yum_output

  - name: Reboot Host if packages were updated
    shell: sleep 2 &amp;&amp; shutdown -r now "Ansible␣updates␣triggered"
    async: 1
    poll: 0
    ignore_errors: true
    when: ('"Complete!" in "{{␣yum_output.stdout_lines[-1]␣}}"') or
          ('"Komplett!" in "{{␣yum_output.stdout_lines[-1]␣}}"')

  - name: waiting for access server
    local_action: wait_for
      host={{ inventory_hostname }}
      state=started
      port=22
      delay=30
      timeout=300
      connect_timeout=15
```

Zuerst wird das Kommando zur Aktualisierung der zu den angegebenen RHSA gehörenden Pakete auf den Hosts ausgeführt. Dabei wird die Ausgabe des Kommandos yum auf den Hosts der Variable yum_output [13] zugewiesen. Diese Variable wird im nächsten Schritt ausgewertet, um zu ermitteln, ob Pakete aktualisiert wurden. Ist dies der Fall, wird der Host neugestartet.

Erklärung: Wurden Pakete aktualisiert, steht in der letzten Zeile der YUM-Ausgabe der Ausdruck "Complete!". `"\{\{␣yum\_output.stdout\_lines[-1]␣}}"}` dereferenziert die Variable und liefert die Ausgabe des YUM-Kommandos als Liste zurück. Dabei wird in diesem Fall auf das letzte Element der Liste zugegriffen. Enthält diese Zeile den genannten Ausdruck, wird der Host neugestartet. *Hinweis:* Die zweite Zeile der when-Bedingung aus dem oberen Listing dient der Behandlung einer deutschsprachigen Ausgabe. In diesem Fall wird das Schlüsselwort "Komplett!äusgegeben und in der Variable gespeichert.

Der letzte Task wartet 30 Sekunden ab, um dem Host Zeit für den Neustart zu geben und prüft, ob eine Verbindung hergestellt werden kann. Damit soll sichergestellt werden, dass der Host nach dem Neustart wieder korrekt hochfährt.

Damit sind die Anforderungen aus Punkt 2 und 3 erfüllt. Um auch noch die Anforderung aus Punkt 1 zu erfüllen, setze ich folgendes Wrapper-Skript ein, welches das Playbook zu den definierten Stichtagen ausführen soll:

Listing 6.4: Skript zur Bestimmung der Stichtage

```
#!/bin/sh

DOM=`date +%d`
DOW=`date +%w`

if [ "$DOW" = "2" ] && [ "$DOM" -gt 7 ] && [ "$DOM" -lt 15 ]
then
  ansible-playbook patch_rhel.yml --limit=e-stage
fi

if [ "$DOW" = "2" ] && [ "$DOM" -gt 14 ] && [ "$DOM" -lt 22 ]
then
  ansible-playbook patch_rhel.yml --limit=q-stage
fi

if [ "$DOW" = "2" ] && [ "$DOM" -gt 21 ] && [ "$DOM" -lt 29 ]
then
```

```
18   ansible-playbook patch_rhel.yml --limit=p-stage
19 fi
```

Mit diesem Wrapper-Skript möchte ich dafür sorgen, dass an jedem 2. Dienstag im Monat die RSHA-Aktualisierungen auf den Hosts in der E-Stage, jeden 3. Dienstag in der Q-Stage und jeden 4. Dienstag in der P-Stage installiert werden.

So kann sichergestellt werden, dass die Informationen aus den RHSA alle Hosts erreichen. Gleichzeitig wird den Betreibern der Hosts die Gelegegnheit gegeben, die Aktualisierungen bis zu den genannten Stichtagen selbst einzuspielen. Damit sollte ich auch an Punkt 1 einen Haken dran machen können.

7 Was ich am Ad-hoc-Modus schätze

Schon seit einiger Zeit hilft mir Ansible fast täglich dabei, meine Arbeit leichter zu gestalten. Heute möchte ich euch ganz kurz erzählen, was ich am Ad-hoc-Modus schätze.

Der Ad-hoc-Modus bietet die Möglichkeit, einfache Kommandos parallel auf einer Gruppe von Nodes ausführen zu lassen, ohne zuvor ein Playbook erstellen zu müssen. Ein Ad-hoc-Befehl besitzt z.B. den folgenden Aufbau:

Listing 7.1: Aufbau eines Ansible-Ad-hoc-Befehls

```
ansible [-m module_name] [-a args] [options]
```

Ein einfaches Beispiel aus der Ansible-Dokumentation [12] soll die Anwendung verdeutlichen:

Listing 7.2: Beispiel eines Ansible-Ad-hoc-Befehls

```
# ansible all -m ping -i staging --limit=e-stage
host01.example.com | SUCCESS =&gt; {
"changed": false,
"ping": "pong"
}
host02.example.com | SUCCESS =&gt; {
"changed": false,
"ping": "pong"
}
host03.example.com | SUCCESS =&gt; {
"changed": false,
"ping": "pong"
}
```

Das Schlüsselwort all gibt an, dass das Kommando auf allen Nodes ausgeführt werden soll, welche in der Inventar-Datei enthalten sind. Mit -m ping wird das zu verwendende Ansible-Modul spezifiziert. Da das verwendete Modul keine weiteren Argumente besitzt, findet -a in diesem Beispiel keine Anwendung. Mit der Option -i kann die zu verwendende Inventar-Datei angegeben werden. Lässt man diese Option weg, wird die Standard-Inventar-Datei /etc/ansible/hosts verwendet. Mit der Option –limit=e-stage wird die Ausführung noch weiter eingeschränkt. So wird in diesem Fall das Modul ping nur auf den Nodes der Gruppe e-stage ausgeführt. Das in diesem Beispiel verwendete Inventar besitzt den folgenden Aufbau:

Listing 7.3: Inventory

```
[e-stage]
host01.example.com
host02.example.com
host03.example.com
host06.example.com
host07.example.com

```

```
8  [i-stage]
9  host04.example.com
10
11 [p-stage]
12 host05.example.com
```

7.1 Verknüpfung mit weiteren Kommandos

Selbstverständlich lassen sich Ansible-Ad-hoc-Kommandos auf der Kommandozeile auch weiter verknüpfen. Dies soll an zwei kleinen Beispielen verdeutlicht werden.

7.1.1 Status eines Dienstes prüfen

In diesem ersten Beispiel soll der Status des Dienstes chronyd überprüft werden, ohne den aktuellen Status zu ändern. Dabei soll das Kommando `systemctl status chronyd.service` via Ansible parallel auf den Nodes ausgeführt werden.

Zuvor habe ich mir auf einem Node angesehen, wie die Ansible-Ausgabe in Abhängigkeit vom Dienststatus aussieht (Ausgabe gekürzt):

Listing 7.4: Rückgabewerte von chronyd

```
1  # Der Dienst auf dem Node ist gestartet
2  root@ansible-control-machine>ansible all -m command -a'/usr/bin/systemctl status
       chronyd.service' -i staging -l host01.example.com
3  host01.example.com | SUCCESS | rc=0 >>
4  * chronyd.service - NTP client/server
5  Loaded: loaded (/usr/lib/systemd/system/chronyd.service; enabled; vendor preset:
       enabled)
6  Active: active (running) since Thu 2016-12-15 14:52:02 CET; 19h ago}
7
8  # Der Dienst auf dem Node ist gestoppt
9  root@ansible-control-machine>ansible all -m command -a'/usr/bin/systemctl status
       chronyd.service' -i staging -l host01.example.com
10 host01.example.com | FAILED | rc=3 >>
11 * chronyd.service - NTP client/server
12 Loaded: loaded (/usr/lib/systemd/system/chronyd.service; enabled; vendor preset:
       enabled)
13 Active: inactive (dead) since Fri 2016-12-16 10:04:34 CET; 4s ago
14
15 # Das Paket, welches den Dienst enthaelt ist nicht installiert
16 root@ansible-control-machine>ansible all -m command -a'/usr/bin/systemctl status
       chronyd.service' -i staging -l host01.example.com
17 host01.example.com | FAILED | rc=4 >>
18 Unit chronyd.service could not be found.
```

Anhand der Ausgaben ist zu erkennen, dass Ansible den Task als SUCCESS —"markiert, wenn der Dienst läuft und als FAILED —", wenn der Dienst gestoppt bzw. gar nicht installiert ist. Durch Verknüpfung des Kommandos mit grep kann man sich nun schnell einen Überblick über den Dienststatus auf seinen Rechnern verschaffen:

Listing 7.5: Überprüfung des Status von chronyd

```
1  root@ansible-control-machine>ansible all -m command -a'/usr/bin/systemctl status
       chronyd.service' -i staging --limit=e-stage | grep '| SUCCESS |\|| FAILED
      |'
2  host01.example.com | SUCCESS | rc=0 >>
3  host02.example.com | SUCCESS | rc=0 >>
4  host03.example.com | FAILED | rc=3 >>
5  host06.example.com | SUCCESS | rc=0 >>
6  host07.example.com | SUCCESS | rc=0 >>
```

Anhand der Ausgabe ist leicht zu erkennen, dass der Dienst chronyd auf host03 nicht läuft. Anhand des Return-Codes rc=3 lässt sich weiterhin erkennen, dass das notwendige Paket offensichtlich installiert ist, der Dienst jedoch nicht gestartet wurde. Dies kann nun jedoch schnell durch folgenden Befehl korrigiert werden (Ausgabe gekürzt):

Listing 7.6: Start von chronyd

```
root@ansible-control-machine>ansible host03.example.com -m systemd -a'name=
    chronyd state=started' -i staging
host03.example.com | SUCCESS | >> {
"changed": true,
"name": "chronyd",
"state": "started",
"status": {...}
}
```

Eine erneute Ausführung des ersten Kommandos bestätigt, dass der Dienst nun auch auf dem Node host03 ausgeführt wird.

Listing 7.7: Erneute Überprüfung des Status von chronyd

```
root@ansible-control-machine&gt; ansible all -m command -a'/usr/bin/systemctl
    status chronyd.service' -i staging --limit=e-stage | grep '| SUCCESS |\||
    FAILED |'
host01.example.com | SUCCESS | rc=0 >>
host02.example.com | SUCCESS | rc=0 >>
host03.example.com | SUCCESS | rc=0 >>
host06.example.com | SUCCESS | rc=0 >>
host07.example.com | SUCCESS | rc=0 >>
```

7.1.2 Paketversion überprüfen

In diesem Beispiel möchte ich die installierte Version des Pakets tzdata abfragen. Dies geschieht auf einem einzelnen Host mit dem Kommando `rpm -qi`:

Listing 7.8: Abfrage mit rpm -qi

```
# rpm -qi tzdata
Name : tzdata
Version : 2016i
Release : 1.el7
Architecture: noarch
Install Date: Wed Nov 9 08:47:03 2016
Group : System Environment/Base
Size : 1783642
License : Public Domain
Signature : RSA/SHA256, Fri Nov 4 17:21:59 2016, Key ID 199e2f91fd431d51
Source RPM : tzdata-2016i-1.el7.src.rpm
Build Date : Thu Nov 3 12:46:39 2016
Build Host : ppc-045.build.eng.bos.redhat.com
Relocations : (not relocatable)
Packager : Red Hat, Inc. &lt;http://bugzilla.redhat.com/bugzilla&gt;
Vendor : Red Hat, Inc.
URL : https://www.iana.org/time-zones
Summary : Timezone data
Description :
This package contains data files with rules for various timezones around
the world.
```

Mich interessiert lediglich die zweite Zeile, welche die Version des Pakets enthält. Die frage ich nun wie folgt ab:

Listing 7.9: Abfrage der Version von tzdata mit Ansible

```
root@ansible-control-machine&gt; ansible all -m command -a'/usr/bin/rpm -qi
    tzdata' -i staging --limit=e-stage | grep 'SUCCESS\|Version'
```

```
2  host01.example.com | SUCCESS | rc=0 >>
3  Version : 2016f
4  host02.example.com | SUCCESS | rc=0 >>
5  Version : 2016g
6  host03.example.com | SUCCESS | rc=0 >>
7  Version : 2016i
8  host06.example.com | SUCCESS | rc=0 >>
9  Version : 2016i
10 host07.example.com | SUCCESS | rc=0 >>
11 Version : 2016i
```

Ohne Ansible hätte ich diese Aufgaben entweder mit Iteration in einem kurzen Shell-Skript lösen müssen, oder zuerst ein Kochbuch, Manifest, etc. schreiben, welches dann anschließend ausgeführt werden kann. So wurde mir hingegen einiges an Zeit gespart, die ich für andere Dinge verwenden konnte.

Listings

Internet-Quellen

[1] Jörg Kastning. *Ansible – IT-Automation für Jedermann*. [Letzter Abruf: 17.12.2016]. 2016. URL: http://www.my-it-brain.de/wordpress/ansible-it-automation-fuer-jedermann/.

[2] Jörg Kastning. *Linux-Benutzerkonten mit Ansible verwalten*. [Letzter Abruf: 17.12.2016]. 2016. URL: http://www.my-it-brain.de/wordpress/linux-benutzerkonten-mit-ansible-verwalten/.

[3] Red Hat. *Ansible authorized_key - Adds or removes an SSH authoriAnsible*. en. [Letzter Abruf: 17.12.2016]. URL: http://docs.ansible.com/ansible/authorized_key_module.html.

[4] Red Hat. *Ansible Best Practices – Directory Layout*. en. [Letzter Abruf: 17.12.2016]. URL: http://docs.ansible.com/ansible/playbooks_best_practices.html#directory-layout.

[5] Red Hat. *Ansible: copy – Copies files to remote locations*. [Letzter Abruf: 17.12.2016]. URL: http://docs.ansible.com/ansible/copy_module.html.

[6] Red Hat. *Ansible: cron – Manage cron.d and crontab entries*. en. [Letzter Abruf: 17.12.2016]. URL: http://docs.ansible.com/ansible/cron_module.html.

[7] Red Hat. *Ansible Module: user – Manage user accounts*. [Letzter Abruf: 17.12.2016]. URL: http://docs.ansible.com/ansible/user_module.html.

[8] Red Hat. *Ansible Playbooks*. [Letzter Abruf: 17.12.2016]. URL: http://docs.ansible.com/ansible/playbooks.html.

[9] Red Hat. *Ansible: script – Runs a local script on a remote node after transferring it*. en. [Letzter Abruf: 17.12.2016]. URL: http://docs.ansible.com/ansible/script_module.html.

[10] Red Hat. *AUTOMATION FOR EVERYONE*. en. [Letzter Abruf: 17.12.2016]. URL: https://www.ansible.com/.

[11] Red Hat. *How do I generate crypted passwords for the user module?* en. [Letzter Abruf: 17.12.2016]. URL: http://docs.ansible.com/ansible/faq.html#how-do-i-generate-crypted-passwords-for-the-user-module.

[12] Red Hat. *Introduction To Ad-Hoc Commands*. en. [Letzter Abruf: 17.12.2016]. URL: http://docs.ansible.com/ansible/intro_adhoc.html.

[13] Red Hat. *Registered Variables*. en. [Letzter Abruf: 17.12.2016]. URL: https://docs.ansible.com/ansible/playbooks_variables.html#registered-variables.

[14] Red Hat. *yum_repository - Add and remove YUM repositories*. en. [Letzter Abruf: 17.12.2016]. URL: https://docs.ansible.com/ansible/yum_repository_module.html.

[15] Wikipedia. *About Modules*. [Letzter Abruf: 17.12.2016]. URL: http://docs.ansible.com/ansible/modules.html.

[16] Wikipedia. *Ansible*. [Letzter Abruf: 17.12.2016]. URL: https://de.wikipedia.org/wiki/Ansible.

[17] Wikipedia. *Chef*. [Letzter Abruf: 17.12.2016]. URL: https://en.wikipedia.org/wiki/Chef_(software).

[18] Wikipedia. *Puppet*. [Letzter Abruf: 17.12.2016]. URL: https://de.wikipedia.org/wiki/Puppet_(Software).

[19] Wikipedia. *Salt*. [Letzter Abruf: 17.12.2016]. URL: https://en.wikipedia.org/wiki/Salt_(software).

[20] Wikipedia. *YAML*. [Letzter Abruf: 17.12.2016]. URL: https://de.wikipedia.org/wiki/YAML.