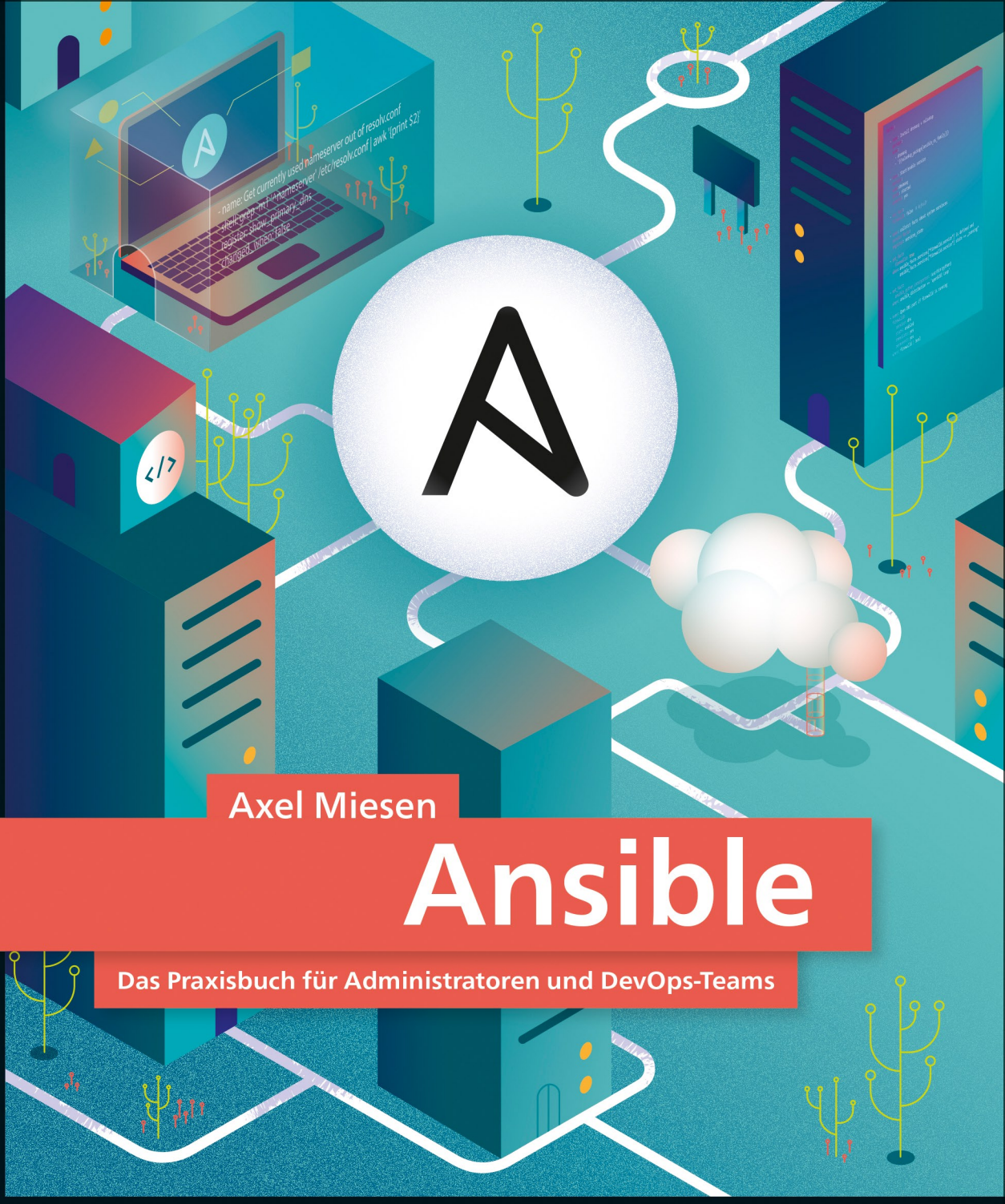

Axel Miesen

Ansible

Das Praxisbuch für Administratoren und DevOps-Teams

- Von den Grundlagen zum orchestrierten Deployment
- Playbooks und Rollen entwickeln
- Best Practices zur Konfigurationsverwaltung, Docker-Integration, Modularisierung, Inventory Management

Alle Beispielprojekte zum Download

Rheinwerk Computing

Kapitel 2

Basiseinrichtung und erstes Inventory-Management

In diesem Abschnitt finden Sie einen Vorschlag, wie Sie Ihre Arbeit mit Ansible auf Verzeichnisebene strukturieren können. Außerdem beschäftigen wir uns mit dem *Inventory* (sozusagen dem Verzeichnis der Target Hosts), das Sie für nahezu jedes Betriebsszenario von Ansible benötigen.

2.1 Verzeichnisstruktur einrichten

Ansible macht uns im Prinzip kaum feste Vorgaben, welche Strukturen wir auf Verzeichnis- und Dateiebene anzulegen haben. Mein Vorschlag dazu ist im Wesentlichen Best Practice und erfüllt zudem folgende Vorgaben:

- Wie schon beim Einrichten der SSH-Public-Key-Authentifizierung praktiziert, wollen wir auf dem Control Host als unprivilegierter User arbeiten.
- Unser gesamtes »Ansible-Projekt« soll sich bis auf Weiteres innerhalb bzw. unterhalb eines einzigen Ordners befinden. Vorteil: Diesen Ordner können Sie jederzeit einfach sichern oder in ein Versionskontrollsystem einchecken, ohne zu befürchten, etwas Wichtiges vergessen zu haben.

Unsere initiale Verzeichnisstruktur (bislang nur bestehend aus leeren Verzeichnissen) soll folgendermaßen aussehen:

```
ansible
|-- inventories
|   `-- devel
|       |-- group_vars
|       `-- host_vars
|-- playbooks
`-- roles
```

Mit diesen Befehlen könnten Sie diese Struktur in Ihrem Heimatverzeichnis anlegen:

```
$ mkdir ~/ansible && cd $_
$ mkdir -p inventories/devel/{host,group}_vars playbooks roles
```

Wir werden diese ganzen Verzeichnisbereiche nach und nach mit Leben füllen; sehen Sie hier dennoch schon eine Kurzübersicht:

- Unter *inventories/* werden wir unser Inventar verwalten (das Hauptthema dieses Kapitels). Die **_vars/*-Ordner dienen zur Parametrisierung von Einzelhosts bzw. Hostgruppen; das werde ich erst in Kapitel 6 thematisieren.
- Unter *playbooks/* werden wir Playbooks ablegen, worauf ich in Kapitel 5 zu sprechen komme.
- Schließlich ist der *roles/*-Ordner für Rollen vorgesehen; dieses Thema muss noch ein klein wenig länger warten und begegnet uns erst in Kapitel 8.

2.2 Grundkonfiguration (»ansible.cfg«)

Da wir Ansible in der Folge mit einigen vom Default abweichenden Einstellungen betreiben wollen, benötigen wir eine Konfigurationsdatei. Diese Datei wird an folgenden Stellen gesucht (der Reihe nach, die erste gewinnt):

1. Inhalt der Umgebungsvariablen `ANSIBLE_CONFIG`
2. *ansible.cfg* im aktuellen Verzeichnis
3. *~/.ansible.cfg*
4. */etc/ansible/ansible.cfg*

Gemäß unseren Vorgaben nutzen wir Möglichkeit Nr. 2. Legen Sie diese erste Version in Ihren *ansible/*-Basisordner:

```
[defaults]
inventory = inventories/devel/inventory
```

Listing 2.1 »ansible.cfg«: erste Version einer Konfigurationsdatei

Ansible wird an dieser Stelle mit einem einfachen INI-Format konfiguriert. Die Erklärung der bislang einzigen Einstellung ist einfach; Sie setzen damit den Pfad zu einer (noch zu erstellenden) Inventory-Datei. Der Default wäre hier */etc/ansible/hosts* gewesen, was nicht zu unseren Vorgaben passt. Solange Sie sich nun im Basisordner befinden, sollte Ansible unsere neue Konfigurationsdatei finden und nutzen:

```
$ ansible --version
ansible 2.9.6
  config file = /home/ansible/ansible/ansible.cfg
  […]
```

Sobald Sie aber Ihr aktuelles Arbeitsverzeichnis ändern, würde Ansible wieder auf die Konfiguration unter */etc* zugreifen, was nicht in unserem Sinne ist. Da es aber auch

nicht immer bequem wäre, alle Aufrufe im Basisordner zu machen, empfiehlt es sich zusätzlich, mit der Umgebungsvariablen `ANSIBLE_CONFIG` stets auf unsere Konfigurationsdatei zu verweisen (Methode 1):

```
$ echo 'export ANSIBLE_CONFIG=~/ansible/ansible.cfg' >>~/.bashrc

$ source ~/.bashrc
```

(Falls Sie eine andere Shell nutzen, müssen Sie hier natürlich eine andere geeignete Startdatei bearbeiten.) Damit wird Ansible nun stets auf unsere soeben erstellte Konfigurationsdatei zugreifen.

Weitere, bereits an dieser Stelle interessante Konfigurationsparameter

Die beiden folgenden Konfigurationsparameter sind bereits an dieser Stelle von Interesse:

- **`log_path`**
  Falls gewünscht, können Sie Ansible mit der Direktive `log_path` dazu veranlassen, alle Aktionen mitzuprotokollieren. Die Logdatei ist frei wählbar, nur das Verzeichnis, das die Logdatei enthalten soll, muss existieren. Wenn also z. B. das Verzeichnis *~/logs/* existiert, könnten Sie folgende Einstellung vornehmen:

  ```
  # [defaults]
  log_path = ~/logs/ansible.log
  ```

  Listing 2.2 »ansible.cfg«: Ausschnitt

- **`private_key_file`**
  Wenn der Dateiname Ihres privaten SSH-Schlüssels vom Default abweicht, können Sie diesen via Konfiguration bekanntmachen, um sich die ständige Angabe mittels `--private-key` auf der Kommandozeile zu ersparen:

  ```
  # [defaults]
  private_key_file = ~/.ssh/ansible_key
  ```

  Listing 2.3 »ansible.cfg«: Ausschnitt

2.3 Erstellen und Verwalten eines statischen Inventorys

Um Ansible mitzuteilen, um welche Hosts es sich in Zukunft kümmern soll, können Sie im einfachsten Fall mit einer *Inventory-Datei* arbeiten (auch als *statisches Inventory* bezeichnet). Unser erstes Ziel soll sein, mittels einer solchen Datei den beliebten und wichtigen Ping-Test zu ermöglichen:

```
$ ansible all -m ping
```

Probieren Sie es gern jetzt schon aus; es kann natürlich nicht funktionieren, da Ansible keine Idee hat, welche Hosts mit »all« gemeint sind.

In unserer *ansible.cfg* haben wir ja schon den Pfad zur Inventory-Datei festgelegt (*inventories/devel/inventory*). Hier folgt ein exemplarischer Inhalt, abgestimmt auf unsere Testumgebung:

```
[test_hosts]
debian
centos
suse    ansible_user=root
ubuntu

[test_hosts:vars]
ansible_python_interpreter=/usr/bin/python3
ansible_ssh_common_args='-o StrictHostKeyChecking=no'
```

Listing 2.4 »inventories/devel/inventory«

Die verwendete Syntax ist nicht ganz klar zu benennen; sagen wir einfach einmal, dass es sich um eine Art INI-Format handelt. Aber davon abgesehen, sollte der Inhalt eigentlich recht gut verständlich sein:

- Mit Namen in eckigen Klammern können Sie Gruppen definieren (in diesem Beispiel: test_hosts). Es besteht keine Pflicht, Hosts in Gruppen zu organisieren, aber es ist sehr üblich und nützlich. Eine spezielle Gruppe namens all, die alle Hosts enthält, ist übrigens stets automatisch vorhanden und muss nicht eigens von Ihnen definiert werden.
- Einzelhosts können mit zusätzlichen »Schlüssel=Wert«-Zuweisungen parametrisiert werden. Wir haben das für die Maschine suse getan, weil dort exemplarisch ein abweichendes Zielaccount verwendet wird.
- Hostgruppen können mit dem Zusatz :vars ebenfalls parametrisiert werden (hier: test_hosts:vars). Als exemplarische Anwendung haben wir Ansible durchgängig die Verwendung von Python 3 vorgeschrieben und für alle Hosts unserer Testgruppe das strikte SSH-Host-Key-Checking deaktiviert, da Vagrant-Hosts dazu neigen, öfter mal mit neuen Keys ausgestattet zu werden. In unserer Testumgebung können wir auf Ereignisse wie

  ```
  The authenticity of host '[…]' can't be established.
  ECDSA key fingerprint is 8a:a3:a5:43:c6:e4:6c:11:3a:c9:2b:97:94:23:40:c9.
  Are you sure you want to continue connecting (yes/no)?
  ```

  oder


```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
```

  getrost verzichten; im Produktivbetrieb rate ich natürlich strengstens von dieser Einstellung ab (bzw. sollte sie nur verwendet werden, wenn Sie sich über die Implikationen völlig im Klaren sind)! Es gäbe sogar die Möglichkeit, in der *ansible.cfg* mit der Direktive host_key_checking = False zu arbeiten, aber dazu rate ich noch viel weniger, da das Checking dann gleich für *alle* Hosts deaktiviert ist und eine solche Einstellung leicht in eine Produktivumgebung »hineinrutschen« könnte.

- Sollten Sie übrigens in Ihrer Umgebung mit der untypischen UNIX-Passwort-Authentifizierung arbeiten, müssten Sie alle Inventory-Hosts noch mit dem zusätzlichen Parameter ansible_password=<LOGIN_PASSWORT> versehen.

Machen Sie nun den Ping-Test – er sollte für alle betrachteten Ansible-Versionen komplett »auf Grün« sein:

```
$ ansible all -m ping
debian | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
ubuntu | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
suse | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
centos | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Die Reihenfolge der Server-Ausgaben ist hier nicht beeinflussbar und wegen paralleler Abarbeitung gewissermaßen zufällig. Dazu aber später mehr.

Python-Probleme?

Falls Sie wider Erwarten noch Probleme mit der zu verwendenden Python-Version haben, dann schauen Sie bitte mal in die offizielle Dokumentation unter *http://docs.ansible.com/ansible/reference_appendices/interpreter_discovery.html*. Ab Ansible 2.8 gibt es hier recht differenzierte Einstellungsmöglichkeiten.

2.4 Inventory-Aliase

Bislang gingen wir davon aus, dass die Hosts im Inventory über ihre dort verzeichneten Namen erreichbar sind; d.h., ein `server57.example.org` im Inventory wäre auch für **`ssh`**, **`ping`** und viele andere Tools mit diesem Namen erreichbar.

Das muss aber nicht der Fall sein. Vielleicht haben Ihre Target Hosts keine zugeordneten Namen oder es gibt zwar welche, aber sie gefallen Ihnen einfach nicht. Das können Sie alles mit *Aliasen* (also alternativen Namen) im Inventory ausgleichen; bspw.:

```
s1  ansible_host=192.168.150.10
s2  ansible_host=192.168.150.20
s3  ansible_host=192.168.150.30
s4  ansible_host=192.168.150.40
```

Listing 2.5 »inventory«: Aliase für IP-Adressen

oder:

```
s1  ansible_host=debian.example.org
s2  ansible_host=centos.example.org
s3  ansible_host=suse.example.org
s4  ansible_host=ubuntu.example.org
```

Listing 2.6 »inventory«: Aliase für Namen

Dabei müssen die `ansible_host`-Namen natürlich wieder auflösbar sein. Für den Fall, dass Ihre Hosts nicht auf Standard-Port 22 erreichbar sind, steht Ihnen natürlich auch der Parameter `ansible_port` zur Verfügung:

```
s1  ansible_host=192.168.150.10  ansible_port=2201
s2  ansible_host=192.168.150.20  ansible_port=2202
s3  ansible_host=192.168.150.30  ansible_port=2203
s4  ansible_host=192.168.150.40  ansible_port=2204
```

Listing 2.7 »inventory«: Aliase mit zusätzlicher Port-Angabe

Es gibt durchaus noch mehr solcher `ansible_*`-Parameter, aber für den Moment sollten Sie gut versorgt sein. Wer Interesse hat, mag gern einen Blick auf die Webseite *http://docs.ansible.com/ansible/intro_inventory.html*, Abschnitt »List of Behavioral Inventory Parameters«, werfen.

2.5 Jenseits von Ping

Ein funktionierender Ping-Test ist ein wichtiger Schritt auf dem Weg zur Basiseinrichtung, aber da er auf den Zielsystemen keinerlei Admin-Rechte erfordert, kann er

leider *nicht* zeigen, ob auch die für alle ernsthafteren Aufgaben erforderliche Rechteerhöhung funktioniert.

Und natürlich tut sie das momentan nicht. Lassen Sie sich als Beweis einmal die erste Zeile der Datei */etc/shadow* auf allen Zielsystemen ausgeben; eine Anforderung, für die Sie zweifellos Root-Rechte benötigen:

```
$ ansible all -a "head -1 /etc/shadow"
debian | FAILED | rc=1 >>
head: cannot open '/etc/shadow' for reading: Permission denied

ubuntu | FAILED | rc=1 >>
head: cannot open '/etc/shadow' for reading: Permission denied

centos | FAILED | rc=1 >>
head: cannot open '/etc/shadow' for reading: Permission denied

suse | CHANGED | rc=0 >>
bin:!:18332::::::
```

Nur die SUSE-Maschine liefert das gewünschte Ergebnis, weil wir dort ja ein direktes Root-SSH zur Verfügung haben. Lassen Sie uns erst noch einmal festhalten, welche Methoden zur Rechteerhöhung auf unseren Testmaschinen zur Verfügung stehen:

Host	Methode
`debian`	sudo
`centos`	su
`suse`	direktes Root-SSH möglich
`ubuntu`	sudo (NOPASSWD)

Eine Möglichkeit, diese ganzen Anforderungen zu erfüllen, zeigt die folgende Inventory-Datei. Ersetzen Sie bitte Ihre bestehende Version durch diese hier:

```
[test_hosts]
debian   ansible_host=192.168.150.10
centos   ansible_host=192.168.150.20
suse     ansible_host=192.168.150.30
ubuntu   ansible_host=192.168.150.40

[debian_hosts]
debian
```

```
[centos_hosts]
centos

[suse_hosts]
suse

[ubuntu_hosts]
ubuntu

########## Parametrisierung: ##########

[test_hosts:vars]
ansible_python_interpreter=/usr/bin/python3
ansible_ssh_common_args='-o StrictHostKeyChecking=no'

[debian_hosts:vars]
ansible_user=ansible
ansible_become=yes
ansible_become_pass=ansible

[centos_hosts:vars]
ansible_user=ansible
ansible_become=yes
ansible_become_method=su
ansible_become_pass=ansible

[suse_hosts:vars]
ansible_user=root

[ubuntu_hosts:vars]
ansible_user=ansible
ansible_become=yes
```

Listing 2.8 »inventories/devel/inventory«

Wir haben etwas umstrukturiert und weitere Gruppen eingeführt, da diese übersichtlicher zu parametrisieren sind als Einzelhosts. Wie Sie sehen, ist es natürlich auch kein Problem, dass Hosts in unterschiedlichen Gruppen auftauchen.

Als Erstes fallen Ihnen vielleicht die Angaben der IP-Adressen auf. Das ist eine reine Absicherung für später, wenn unser Inventory vielleicht einmal in Umgebungen verwendet wird, in denen es unsere Möglichkeit der Namensauflösung nicht gibt (Kapitel 10, »Webinterfaces«).

Die Rechteerhöhung (unser eigenliches Anliegen) wird mit den folgenden Parametern konfiguriert:

- **`ansible_become`**
  Der Hinweis an Ansible, dass auf dem Zielhost eine Rechteerhöhung stattfinden muss. Der Default ist `no`.
- **`ansible_become_method`**
  Die dazu zu verwendende Methode. Der Default ist `sudo`.
- **`ansible_become_user`**
  Das Zielaccount des Identitätwechsels. Der Default ist `root`, was in einem solchen Kontext allermeistens passt. Deswegen taucht dieser Parameter in obigem Beispiel auch überhaupt nicht auf.
- **`ansible_become_pass`**
  Das zur Rechteerhöhung benötigte Passwort.

Zu den Passwörtern gibt es natürlich noch etwas zu bemerken. Im Testbetrieb geht es ja möglicherweise in Ordnung, Klartextpasswörter in einer Konfigurationsdatei vorzuhalten; im Produktivbetrieb wahrscheinlich eher nicht. Welche Alternativen hätten Sie hier?

- Sie könnten sich stattdessen mit der Option `--ask-become-pass / -K` bei jedem Aufruf eines Ansible-Kommandos prompten lassen.
- Sie könnten zumindest im Falle von `sudo` per Konfiguration auf den Target Hosts auf die Passworteingabe verzichten:

  ```
  # Allow members of group sudo to execute any command
  %sudo   ALL=(ALL:ALL) NOPASSWD: ALL
  ```

  Listing 2.9 »/etc/sudoers«: Verzicht auf Passwort-Authentifizierung

Beides will nicht so recht überzeugen, und für `su` hätten Sie immer noch gar keine automatische Lösung. Ich kann Sie letztlich an dieser Stelle nur auf später vertrösten, denn wenn Sie bald mehr über Ansible wissen, können Sie Hosts oder Gruppen durchaus auch an anderen Stellen parametrisieren und diese Stellen bei Bedarf auch verschlüsseln. Informationen dazu finden Sie in Abschnitt 11.1.

In unserem Testlabor sind wir mit dem aktuellen Inventory jedenfalls erst einmal für alle weiteren (administrativen) Aktionen gut aufgestellt:

```
$ ansible all -a "head -1 /etc/shadow"
debian | CHANGED | rc=0 >>
root:!:18333:0:99999:7:::

suse | CHANGED | rc=0 >>
bin:!:18332::::::
```

```
ubuntu | CHANGED | rc=0 >>
root:!:18352:0:99999:7:::

centos | CHANGED | rc=0 >>
root:$6$wOrl./pGMSOmXY$qD14wtpRzgtOdXI2PJ[…]
```

2.6 Alternative Inventorys

Ich sollte noch klären, warum die Inventory-Datei vergleichsweise tief in der Verzeichnishierarchie versteckt wurde. So was wie *~/ansible/hosts.txt* hätte es doch auch getan, oder?

Selbstverständlich, aber mit dem vorliegenden Ansatz können Sie viel leichter *unterschiedliche* Inventorys mit zugehöriger Parametrisierung verwalten. Möglicherweise sieht Ihre Struktur schon bald wie diese hier aus:

```
ansible
|-- inventories
|   |-- devel
|   |   |-- group_vars
|   |   |-- host_vars
|   |   `-- inventory
|   |-- production
|   |   |-- group_vars
|   |   |-- host_vars
|   |   `-- inventory
|   `-- staging
|       |-- group_vars
|       |-- host_vars
|       `-- inventory
|-- […]
```

Natürlich kann es nur ein Default-Inventory geben (das Sie in der *ansible.cfg* festgelegt haben), aber bei nahezu jedem Ansible-Kommando können Sie mit dem Schalter `-i` ein alternatives Inventory angeben:

```
$ ansible all -i inventories/production/inventory -m ping
```


Anmerkung

Erinnern Sie sich noch an die allerersten Testaufrufe, als Sie noch keinerlei Konfiguration hatten? Etwa:

```
$ ansible all -i debian, -m ping
```

Der Schalter `-i` erlaubt es auch, die Liste der Zielhosts komma-separiert direkt auf der Kommandozeile anzugeben. Wenn man aber nur einen einzigen Host angeben möchte, würde Ansible ohne das Komma am Ende den Hostnamen für einen Dateinamen halten, und der Aufruf würde natürlich fehlschlagen.

Ich möchte das Thema »Inventory« an dieser Stelle erst einmal beenden, weil wir noch andere Dinge vor uns haben und uns hier nicht zu sehr in Details verlieren wollen.

Ich werde aber in Kapitel 7 noch einmal darauf zurückkommen, weil es durchaus noch einiges dazu zu sagen gibt. Beispielsweise haben Sie sich bestimmt schon gedacht, dass es auch ein *dynamisches* Inventory geben muss, da ich unseres ja als »statisch« bezeichnet habe. Oder Sie haben sich gefragt, ob es auch Gruppen von Gruppen gibt. All das werden wir noch klären.

Kapitel 7

Inventory-Management: fortgeschrittene Methoden

In Kapitel 2 haben wir statische Inventorys betrachtet und mit Absicht noch einige Fragen zum Thema Inventory-Management offen gelassen. Nun ist ein guter Zeitpunkt, um noch einmal auf Inventorys zurückzukommen.

7.1 Das Kommando »ansible-inventory«

Lassen Sie uns mit einem weiteren nützlichen Tool aus der Ansible-Suite beginnen: `ansible-inventory`. Mit diesem Kommando können Sie Informationen über Ihr Inventory gewinnen und die Parametrisierung von Hosts darstellen.

Versuchen Sie einmal Aufrufe wie:

```
Übersicht (ASCII-Grafik); ohne und mit Gruppenparametrisierung:
ansible-inventory --graph
ansible-inventory --graph --vars

Das Gleiche, eingeschränkt auf eine Gruppe:
ansible-inventory --graph <GROUPNAME>
ansible-inventory --graph --vars <GROUPNAME>

Übersicht (JSON bzw. YAML) mit Parametrisierung:
ansible-inventory --list
ansible-inventory --list --yaml

Übersicht Parametrisierung von einzelnen Hosts:
ansible-inventory --host <HOSTNAME>
ansible-inventory --host --yaml <HOSTNAME>
```

Hier eine exemplarische Ausgabe:

```
$ ansible-inventory --graph
@all:
  |--@centos_hosts:
```

```
|  |--centos
|--@debian_hosts:
|  |--debian
|--@suse_hosts:
|  |--suse
|--@test_hosts:
|  |--centos
|  |--debian
|  |--suse
|  |--ubuntu
|--@ubuntu_hosts:
|  |--ubuntu
|--@ungrouped:
```

7.2 Verschachtelte Gruppen

Anders als bei Benutzergruppen in Linux können Gruppen in einem Ansible-Inventory durchaus verschachtelt sein, d. h., Sie können Gruppen von Gruppen definieren.

Eine solche »Supergruppe« definieren Sie im statischen Inventory, indem Sie an den Gruppennamen das Suffix »:children« anhängen:

```
[debian_hosts]
debian

[centos_hosts]
centos

[suse_hosts]
suse

[ubuntu_hosts]
ubuntu

[apt_hosts:children]
debian_hosts
ubuntu_hosts

[rpm_hosts:children]
centos_hosts
suse_hosts
```

Listing 7.1 »inventory«: Beispiel für verschachtelte Gruppen

Sie sehen in diesem Beispiel die zwei Gruppen `apt_hosts` und `rpm_hosts`, die dann jeweils die Mitglieder der aufgeführten einfachen Gruppen enthalten. Supergruppen können natürlich mit `:vars` genauso parametrisiert werden, wie Sie das von den normalen Gruppen bereits kennen.

Testen wir, ob es auch funktioniert:

```
$ ansible-inventory --graph apt_hosts
@apt_hosts:
  |--@debian_hosts:
  |  |--debian
  |--@ubuntu_hosts:
  |  |--ubuntu
```

Die Verschachtelungstiefe ist übrigens nicht begrenzt; wenn es für Sie zweckdienlich ist, können Sie Gruppen von Gruppen von Gruppen von ... definieren.

7.3 »On-the-fly«-Inventorys erstellen mit »add_host«

Mitunter möchten Sie mit Ansible Target Hosts managen, die weder in einem statischen noch in einem dynamischen Inventory verzeichnet sind. Beispielsweise, weil die Systeme eben erst provisioniert wurden und Sie kein adäquates dynamisches Inventory-Handling zur Verfügung haben.

Als Alternative gibt es in Ansible die Möglichkeit, mit dem `add_host`-Modul Hosts und Gruppen zu einem temporären Inventory hinzuzufügen, um in einem späteren Play mit diesen zu arbeiten. Dieses Inventory befindet sich lediglich im Hauptspeicher und ist nach dem Playbook-Lauf wieder vergessen.

In folgendem Beispiel steckt der Task im ersten Play zwei unserer schon vorhandenen Hosts unter neuem Namen in eine Gruppe namens `temp_group`. Das zweite Play arbeitet dann mit diesen Hosts:

```
---
- hosts: localhost
  tasks:
    - name: Hosts bekanntmachen
      add_host:
        hostname: "{{item.ip}}"
        groups: temp_group
        ansible_become:        yes
        ansible_become_method: "{{item.become_method |default(omit)}}"
        ansible_become_pass:   "ansible"
        ansible_ssh_common_args: '-o StrictHostKeyChecking=no'
```

```
      with_items:
        - ip: 192.168.150.10
        - ip: 192.168.150.20
          become_method: su

- hosts: temp_group
  tasks:
    - debug: msg="Hallo {{inventory_hostname}}"
    - command: head -1 /etc/shadow
```

Listing 7.2 »add_host.yml«

Um ganz sicher zu beweisen, dass unser vorhandenes statisches Inventory das Verhalten nicht beeinflusst, sorgen wir beim Aufruf des Playbooks für ein leeres Inventory:

```
$ ansible-playbook -i, add_host.yml
[WARNING]: provided hosts list is empty, only localhost is available.
Note that the implicit localhost does not match 'all'

PLAY [localhost] ***************************************************************

TASK [Hosts bekanntmachen] *****************************************************
changed: [localhost] => (item={'ip': '192.168.150.10'})
changed: [localhost] => (item={'ip': '192.168.150.20', 'become_method': 'su'}

PLAY [temp_group] **************************************************************

TASK [debug] *******************************************************************
ok: [192.168.150.10] => {
    "msg": "Hallo 192.168.150.10"
}
ok: [192.168.150.20] => {
    "msg": "Hallo 192.168.150.20"
}

TASK [command] *****************************************************************
changed: [192.168.150.10]
changed: [192.168.150.20]

PLAY RECAP *********************************************************************
192.168.150.10             : ok=2    changed=1    unreachable=0    failed=0
192.168.150.20             : ok=2    changed=1    unreachable=0    failed=0
localhost                  : ok=1    changed=1    unreachable=0    failed=0
```


7.4 Dynamische Gruppen mit »group_by«

Ein weiteres, sehr mächtiges Feature von Ansible ist das `group_by`-Modul, mit dem Sie während eines Playbook-Laufs dynamische Gruppen bilden können. Wozu dient das? Stellen Sie sich vor, Sie haben in Ihrem Inventory eine Gruppierung nach Standorten oder Themengebieten (Mailserver, Webserver etc.) vorgenommen, brauchen nun aber eine Gruppierung nach Betriebssystemen (Debian, Suse ...).

Sofern es Ihnen genügt, dass diese neue Gruppierung nicht dauerhaft im Inventory verzeichnet ist, sondern nur dynamisch bei Bedarf zur Verfügung steht, kommen Sie mit `group_by` schnell zum Ziel. Im Prinzip können Sie alle bekannten Facts benutzen, um eine Gruppeneinteilung vorzunehmen.

Sehen Sie im folgenden Playbook aus Listing 7.3, wie im ersten Play eine Gruppierung nach Betriebssystemfamilien gebildet wird und in weiteren Plays die Hosts gruppenweise abgearbeitet werden:

```
---
- name: Gruppieren nach Systemfamilien
  hosts: all

  tasks:
    - group_by: key=os_{{ ansible_os_family }}


- name: Alle aus der Debian-Familie abarbeiten
  hosts: os_Debian

  tasks:
    - debug: msg="Hallo {{ inventory_hostname }}"


- name: Und nun der Rest
  hosts: all:!os_Debian

  tasks:
    - debug: msg="Hallo {{ inventory_hostname }}"
```

Listing 7.3 »group_by.yml«: dynamische Gruppierung nach Betriebssystemfamilien

Und so sieht der Ablauf aus:

```
$ ansible-playbook group_by.yml
[…]
PLAY [Alle aus der Debian-Familie abarbeiten] ******************************
```

```
TASK [debug] *******************************************************************
ok: [debian] => {
    "msg": "Hallo debian"
}
ok: [ubuntu] => {
    "msg": "Hallo ubuntu"
}

PLAY [Und nun der Rest] ********************************************************

TASK [debug] *******************************************************************
ok: [centos] => {
    "msg": "Hallo centos"
}
ok: [suse] => {
    "msg": "Hallo suse"
}
[…]
```

Natürlich müssen Sie die ganzen entstandenen Gruppen von Hosts nicht alle verwenden (es hätte ja auch noch `os_Suse` und `os_RedHat` gegeben). Ein `group_by` stellt Gruppen nur zur Verfügung – was Sie damit anfangen, ist allein Ihnen überlassen.

Listing 7.4 zeigt noch eine andere Idee – eine Gruppierung nach der Major-Kernelversion:

```
---
- name: Gruppieren nach Major-Kernelversion
  hosts: all
  tasks:
    - group_by: key=kernel_{{ ansible_kernel | regex_search('^\d+') }}


- name: Alle Hosts mit Kernel 4.x
  hosts: kernel_4
  tasks:
    - debug: msg="Hier läuft noch ein 4er-Kernel ({{ansible_kernel}})"


- name: Alle Hosts mit Kernel 5.x
  hosts: kernel_5
  tasks:
    - debug: msg="Hier läuft bereits ein 5er-Kernel ({{ansible_kernel}})"
```

Listing 7.4 »group_by_kernelversion.yml«

Der Jinja-Filter `regex_search` ist vergleichbar mit einem `grep -o` auf der Kommandozeile: Nur der Teil des Inputs, der auf den regulären Ausdruck passt, wird zurückgegeben.

Und da der Ausdruck `'^\d+'` einfach bedeutet »Finde eine oder mehr Ziffern am Anfang des Textes«, bekommen wir damit die Major-Version des Kernels geliefert und können sie zur Gruppeneinteilung verwenden:

```
$ ansible-playbook group_by_kernelversion.yml
[…]
PLAY [Alle Hosts mit Kernel 4.x] ***********************************************

TASK [debug] *******************************************************************
ok: [debian] => {
    "msg": "Hier läuft noch ein 4er-Kernel (4.19.0-8-amd64)"
}
ok: [suse] => {
    "msg": "Hier läuft noch ein 4er-Kernel (4.12.14-lp151.28.36-default)"
}
ok: [centos] => {
    "msg": "Hier läuft noch ein 4er-Kernel (4.18.0-147.3.1.el8_1.x86_64)"
}

PLAY [Alle Hosts mit Kernel 5.x] ***********************************************

TASK [debug] *******************************************************************
ok: [ubuntu] => {
    "msg": "Hier läuft bereits ein 5er-Kernel (5.4.0-26-generic)"
}
[…]
```

7.5 Dynamische bzw. externe Inventorys

Unserem wohlbekannten statischen Inventory sind mitunter Grenzen gesetzt. Wenn die Menge der Target Hosts stark fluktuiert und/oder diese sowieso schon an anderer Stelle inventarisiert sind, dann kann die (zusätzliche) Pflege eines statischen Ansible-Inventorys sehr aufwendig und fehleranfällig werden. Abhilfe schafft hier Ansibles Fähigkeit, *dynamische* bzw. *externe Inventorys* einzubinden.

Dazu gibt es grundsätzlich zwei Methoden:

1. Inventory Scripts (die »klassische« Methode)
2. Inventory-Plugins (die neuere und empfohlene Methode)

Bei beiden Methoden können Sie entweder auf bereits existierende Lösungen zurückgreifen oder selbst neue Lösungen entwickeln (sprich: programmieren). Der vielleicht entscheidende Unterschied ist, dass Sie ein Inventory-Skript in jeder beliebigen Programmiersprache verfassen können, ein Inventory-Plugin jedoch nur in Python.

Da ich persönlich seit über 20 Jahren alle Programmieraufgaben mit Perl erledige, zu Python jedoch nie einen rechten Zugang gefunden habe (es mag mich einfach nicht), werde ich für Methode 1 nur mit einem Beispiel in Perl dienen können. Methode 2 untersuchen wir ausführlicher am Beispiel dreier Plugins:

- Amazon EC2
- Hetzner Cloud
- VMware

Der Einstiegspunkt der offiziellen Dokumentation in diese Thematik ist übrigens *http://docs.ansible.com/ansible/intro_dynamic_inventory.html*.

7.5.1 Beispiel: ein Inventory-Skript in Perl

Angenommen, Sie verwalten Ihre Hosts ohnehin schon in einer Datenbank. Ich habe hier einmal eine »Tabelle« in Form einer CSV-Datei nachgebaut (die erste Zeile definiert die Spaltennamen und damit für uns auch die Bedeutung):

```
host:user:method:password
debian:ansible:sudo:ansible
centos:ansible:su:ansible
suse:root::
ubuntu:ansible:sudo:
```

Listing 7.5 »hosts.csv«

Daraus müssen wir jetzt einen JSON-Output erzeugen, der in wesentlichen Teilen so aussieht wie die momentane Ausgabe von `ansible-inventory --list`. (Genauer ist das beschrieben unter: *http://docs.ansible.com/ansible/developing_inventory.html#developing-inventory-scripts*.)

Das folgende Perl-Skript aus Listing 7.6 ist sogar etwas aufwendiger als eigentlich nötig, da die CSV-Datei mit einem »Datenbanktreiber« verarbeitet wird. Dadurch würde sich am Skript aber auch nicht viel ändern, wenn tatsächlich eine echte SQL-Datenbank eingesetzt würde.

```
#!/usr/bin/perl
use strict;
use warnings;
use DBI;
```



```
use JSON;
use FindBin;

# Konnektieren der Datenquelle ( = CSV-Datei )
my $dbh = DBI->connect ("dbi:CSV:f_dir=${FindBin::Bin}", undef, undef, {
    f_encoding       => "utf8",
    csv_eol          => "\n",
    csv_sep_char     => ":",
});
$dbh->{csv_tables}{hosts}  = { file => "hosts.csv"  };


# Grundeinstellungen:
my $group    = "test_hosts";
my $python   = "/usr/bin/python3";
my $ssh_args = "-o StrictHostKeyChecking=no";


# Grundgerüst der Ergebnis-Struktur in Perl:
my $info = {
    _meta => {
        hostvars => {}
    },
    $group => {
        hosts => []
    }
};


# Abfrage aller Daten:
my $sth = $dbh->prepare("SELECT host, user, method, password FROM hosts");
$sth->execute;


# Verarbeiten der Daten und Befüllen der Struktur:
while (my ($host, $user, $method, $pass) = $sth->fetchrow_array) {
    push @{ $info->{$group}{hosts} }, $host;

    $info->{_meta}{hostvars}{$host}{ansible_user}
        = $user;

    my $ansible_become = $method? "yes" : "no";
```

```
    $info->{_meta}{hostvars}{$host}{ansible_become}
        = $ansible_become;

    $info->{_meta}{hostvars}{$host}{ansible_become_method}
        = $method if $method;

    $info->{_meta}{hostvars}{$host}{ansible_become_pass}
        = $pass if $pass;

    $info->{_meta}{hostvars}{$host}{ansible_python_interpreter}
        = $python;

    $info->{_meta}{hostvars}{$host}{ansible_ssh_common_args}
        = $ssh_args;
}


# Ausgabe im JSON-Format:
my $json = JSON->new;
print $json->pretty->encode( $info );
```

Listing 7.6 »dyninv.pl«

Legen Sie beide Dateien unter *inventories/devel* ab. Bevor Sie das Programm wie beabsichtigt nutzen können, müssen Sie aber noch zwei Dinge erledigen:

1. Als Admin die benötigten Perl-Module installieren. Auf einem Debian-System reicht dazu ein

   ```
   # apt install libdbd-csv-perl libjson-perl
   ```

2. Das Programm ausführbar machen:

   ```
   $ chmod +x dyninv.pl
   ```

Nun sollten Sie es starten können:

```
$ ./dyninv.pl
{
   "test_hosts" : {
      "hosts" : [
         "debian",
         "centos",
         "suse",
         "ubuntu"
      ]
   },
```



```
"_meta" : {
   "hostvars" : {
      "debian" : {
         "ansible_user" : "ansible",
         "ansible_ssh_common_args" : "-o StrictHostKeyChecking=no",
         "ansible_python_interpreter" : "/usr/bin/python3",
         "ansible_become_pass" : "ansible",
         "ansible_become" : "yes",
         "ansible_become_method" : "sudo"
      },
      "centos" : {
         "ansible_user" : "ansible",
         "ansible_ssh_common_args" : "-o StrictHostKeyChecking=no",
         "ansible_python_interpreter" : "/usr/bin/python3",
         "ansible_become_pass" : "ansible",
         "ansible_become" : "yes",
         "ansible_become_method" : "su"
      },
      "ubuntu" : {
         "ansible_user" : "ansible",
         "ansible_ssh_common_args" : "-o StrictHostKeyChecking=no",
         "ansible_become" : "yes",
         "ansible_become_method" : "sudo",
         "ansible_python_interpreter" : "/usr/bin/python3"
      },
      "suse" : {
         "ansible_ssh_common_args" : "-o StrictHostKeyChecking=no",
         "ansible_become" : "no",
         "ansible_user" : "root",
         "ansible_python_interpreter" : "/usr/bin/python3"
      }
   }
}
```

Entscheidend ist aber, dass es auch im Ansible-Kontext wie gewünscht funktioniert. Führen Sie dazu einige Ihnen bereits bekannte Tests durch, aber dieses Mal mit dem dynamischen Inventory:

```
$ ansible-inventory -i ./dyninv.pl --list
[…]

$ ansible all -i ./dyninv.pl -m ping
[…]
```

```
$ ansible all -i ./dyninv.pl -a "head -1 /etc/shadow"
[…]
```

7.5.2 Verwenden von Inventory-Plugins

Wenn Sie ein Inventory spezifizieren (z. B. mit dem `-i`-Schalter), so haben Sie noch viel mehr Möglichkeiten als die bislang gezeigten. Über sogenannte *Inventory-Plugins* können beliebige Quellen eingebunden werden; die eben beschriebene Inventory-Skript-Variante ist intern letztlich auch durch ein Plugin realisiert.

Per Default versteht Ansible auch das Laden eines Plugins über eine YAML-Konfigurationsdatei mit gewissen Eigenschaften, die in der Dokumentation des jeweiligen Plugins beschrieben sind.

Die allgemeine, offizielle Dokumentation zum Thema ist *http://docs.ansible.com/ansible/plugins/inventory.html*. Ganz am Ende finden Sie dort auch eine Liste der aktuell angebotenen Plugins. In der Folge sehen wir uns einige Plugins etwas genauer an. Alle diese Inventory-Plugins werden auch mit einer Offline-Dokumentation geliefert. Diese lesen Sie bei Bedarf mittels:

```
$ ansible-doc -t inventory <PLUGIN_NAME>
```

7.5.3 AWS EC2

Amazons Elastic Compute Cloud (EC2) gehört sicher zu den populärsten IaaS-Plattformen überhaupt (IaaS = Infrastructure as a Service). Wenn Sie dort virtuelle Server betreiben (Amazon nennt das *Instances*), haben Sie sicher ein Interesse daran, das dortige Inventory in Ansible einzubinden.

Anmerkung

Ich thematisiere hier *nicht* das EC2-Management mittels Ansible, wie z. B. das Erstellen neuer Instanzen. Dazu steht eine Vielzahl an `ec2_*`-Modulen zur Verfügung; ich verweise dazu auf die offizielle Ansible-Dokumentation.

Bevor das EC2-Inventory-Plugin ins Spiel kommen kann, müssen zwei Voraussetzungen erfüllt sein:

1. Sie benötigen auf dem Control Host die Python-Module *boto3* und *botocore*. Auf unserem Labor-Host erledigen Sie das als Admin mit:

   ```
   # apt install python3-boto3 python3-botocore
   ```


2. Sie benötigen Zugangsdaten für die EC2-API. Die Username-Passwort-Kombination, mit der Sie möglicherweise gerade in der Amazon Web-Konsole unterwegs sind, ist dafür nicht geeignet.

   Vielmehr müssen Sie über das Identity and Access Management (IAM) einen Benutzer anlegen, und dort bekommen Sie dann eine *Zugriffsschlüssel-ID* und einen *geheimen Zugriffsschlüssel*. Die Berechtigung `AmazonEC2ReadOnlyAccess` ist für den Inventory-Zugriff ausreichend. Sichern Sie sich am Ende des Prozesses die Anmeldeinformationen! Der geheime Zugriffsschlüssel kann später nicht mehr sichtbar gemacht werden; Sie müssten dann den Benutzer erneut anlegen.

Kommen wir nun zur Ansible-Integration. Da wir hier ein neues Inventory verwalten möchten, sollten wir in unserem Inventory-Ordner auch ein neues Unterverzeichnis anlegen, etwa *ec2_demo*:

```
$ cd inventories
$ mkdir ec2_demo && cd ec2_demo
```

In diesem Ordner erstellen Sie dann die benötigte YAML-Konfigurationsdatei. Das Plugin verlangt, dass der Dateiname auf *aws_ec2.(yml|yaml) endet*, also nennen wir sie bspw. *demo.aws_ec2.yml*.

So sieht die Minimalversion für einen ersten Testlauf aus:

```
---
plugin: aws_ec2
aws_access_key: IHRE_ZUGRIFFSSCHLÜSSEL_ID
aws_secret_key: IHR_GEHEIMER_ZUGRIFFSSCHLÜSSEL
```

Listing 7.7 »inventories/ec2_demo/demo.aws_ec2.yml«: Basis-Plugin-Konfiguration

Alternativ zu den Einträgen in der Datei können Sie auch mit den Umgebungsvariablen `AWS_ACCESS_KEY` bzw. `AWS_SECRET_KEY` arbeiten.

Nun sind Sie bereit für einen ersten Test – nach zehn bis 15 Sekunden Wartezeit sehen Sie mit folgendem Aufruf (hoffentlich) Ihr Inventar:

```
$ ansible-inventory -i demo.aws_ec2.yml --list
```

Mit dem Inventory arbeiten

Ein funktionierendes Inventory ist erst die halbe Miete; Sie wollen ja mit den Target-Hosts auch noch etwas tun!

Grundlage dafür ist ein geeignetes und sauberes Tagging Ihrer Maschinen, denn ohne Zusatzinformationen können Sie nicht einmal wissen, welches Betriebssystem auf einem Target installiert ist oder mit welchem Useraccount Sie sich dort anmelden müssen.

Das Tagging von EC2-Instanzen ist eine Wissenschaft für sich, und jeder hat hier eigene Anforderungen. Nehmen wir exemplarisch einmal an, alle Ihre Maschinen haben Tags, wie in Tabelle 7.1 beschrieben.

Key	mögliche Werte
name	srv01, srv02 ...
role	webapp, db ...
os	ubuntu, amazonlinux ...

Tabelle 7.1 Exemplarische Tags für EC2-Instanzen

Dann können Sie das Inventory-Plugin mittels `keyed_groups` anweisen, entsprechende Gruppen zur Verfügung zu stellen:

```
---
plugin: aws_ec2
aws_access_key: IHRE_ZUGRIFFSSCHLÜSSEL_ID
aws_secret_key: IHR_GEHEIMER_ZUGRIFFSSCHLÜSSEL

keyed_groups:
  - prefix: tag
    key: tags
```

Listing 7.8 »demo.aws_ec2.yml«: Konfiguration mit Gruppenbildung

Damit bekämen Sie dann Gruppen wie `tag_os_ubuntu`, `tag_role_webapp` ... – je nachdem, was bei Ihnen so alles vorhanden ist.

Jetzt müssen die Gruppen nur noch parametrisiert werden (welcher User ist zum Login geeignet, welcher SSH-Schlüssel soll verwendet werden ...). Dazu sollten normale *group_vars*-Parameter ausreichen; legen Sie neben Ihrer YAML-Konfigdatei einen Ordner *group_vars* an, und platzieren Sie darin bspw. folgende Dateien:

```
---
ansible_become: yes
ansible_ssh_private_key_file: '~/.ssh/aws_ec2.pem'
```

Listing 7.9 »group_vars/all.yml«

```
---
ansible_user: ubuntu
```

Listing 7.10 »group_vars/tag_os_ubuntu.yml«


```
---
ansible_user: ec2-user
```

Listing 7.11 »group_vars/tag_os_amazonlinux.yml«

Nun können Sie ernsthaft testen (wichtig ist natürlich, dass der zweite Aufruf problemlos läuft):

```
$ ansible -i demo.aws_ec2.yml all -m ping
$ ansible -i demo.aws_ec2.yml all -a "head -1 /etc/shadow"
```

In Abschnitt 7.5.6 beschreibe ich, wie Sie die nahezu unerträglich langen Antwortzeiten mittels Caching verkürzen können.

7.5.4 Hetzner Cloud

Auch die deutsche Firma Hetzner bietet eine sehr gute und flexible Cloud-Lösung; lassen Sie uns einen Blick auf das entsprechende Ansible-Inventory-Plugin werfen.

Anmerkung

Das Management von Objekten in der Hetzner Cloud mittels Ansible ist hier nicht das Thema; ich komme aber in den Abschnitten 12.2 ff. noch darauf zu sprechen.

Bevor das HCloud-Inventory-Plugin ins Spiel kommen kann, müssen zwei Voraussetzungen erfüllt sein:

1. Sie benötigen auf dem Control Host das Python-Modul *hcloud*. Auf unserem Labor-Host erledigen Sie das als Admin mit:

   ```
   # pip3 install hcloud
   ```

2. Sie benötigen Zugangsdaten für die HCloud-API. Die Username-Passwort-Kombination, mit der Sie möglicherweise gerade in der Hetzner-Web-Konsole unterwegs sind, ist dafür nicht geeignet. Vielmehr müssen Sie in der Web-Konsole im entsprechenden Projekt unter SICHERHEIT im Reiter API-TOKENS ein API-Token generieren. Sichern Sie dieses Token, denn Sie können es später nicht mehr sichtbar machen. Sie müssten dann ein neues anlegen.

Kommen wir nun zur Ansible-Integration. Da wir hier ein neues Inventory verwalten möchten, sollten wir in unserem Inventory-Ordner auch ein neues Unterverzeichnis anlegen, etwa *hcloud_demo*:

```
$ cd inventories
$ mkdir hcloud_demo && cd hcloud_demo
```

In diesem Ordner erstellen Sie dann die benötigte YAML-Konfigurationsdatei. Das Plugin verlangt, dass der Dateiname auf *hcloud.(yml|yaml) endet*, also nennen wir sie bspw. *demo.hcloud.yml*. So sieht die Minimalversion für einen ersten Testlauf aus:

```
---
plugin: hcloud
token: IHR_ACCESS_TOKEN
```

Listing 7.12 »inventories/hcloud_demo/demo.hcloud.yml«: Basis-Plugin-Konfiguration

Alternativ zum Eintrag in der Datei können Sie auch mit der Umgebungsvariablen `HCLOUD_TOKEN` arbeiten. Nun sind Sie bereit für einen ersten Test. Mit folgendem Aufruf sehen Sie (hoffentlich) Ihr Inventar:

```
$ ansible-inventory -i demo.hcloud.yml --list
```

Mit dem Inventory arbeiten

Mit dem Hetzner-Inventory lässt sich out of the box schon ziemlich gut arbeiten, denn es enthält bereits nützliche Informationen wie Betriebssystemtyp oder Serverstatus. Nach diesen ganzen Eigenschaften können Sie bei Bedarf gruppieren; zumeist versieht man die Server zusätzlich aber noch mit eigenen Labels, über die Sie dann fachliche Gruppen bilden können:

```
---
plugin: hcloud
token: IHR_ACCESS_TOKEN

keyed_groups:
  - key: image_os_flavor
    prefix: os
  - key: status
    prefix: server_status
  - key: labels
    prefix: label
```

Listing 7.13 »demo.hcloud.yml«: Konfiguration mit Gruppenbildung

Damit bekämen Sie dann Gruppen wie `os_debian`, `server_status_running`, `label_type_worker` ... – je nachdem, was bei Ihnen so alles vorhanden ist.

Jetzt müssen die Gruppen nur noch parametrisiert werden. Der Login-User ist bei Hetzner standardmäßig immer `root`, darüber hinaus müssen Sie nur noch den zu verwendenden SSH-Schlüssel festlegen. Dazu sollten normale *group_vars*-Parameter ausreichen; legen Sie neben Ihrer YAML-Konfigdatei einen Ordner *group_vars* an, und platzieren Sie darin im einfachsten Fall folgende Datei:


```
---
ansible_user: root
ansible_ssh_private_key_file: '~/.ssh/hetzner'
```

Listing 7.14 »group_vars/all.yml«

Den öffentlichen Teil des Schlüssels haben Sie bei Hetzner hinterlegt – Sie haben ja hoffentlich keine Server ohne SSH-Key-Zugang angelegt. Den Dateinamen müssen Sie natürlich Ihrer Situation anpassen.

Nun können Sie ernsthaft testen (wichtig ist natürlich, dass der zweite Aufruf problemlos läuft):

```
$ ansible -i demo.hcloud.yml all -m ping
$ ansible -i demo.hcloud.yml all -a "head -1 /etc/shadow"
```

7.5.5 VMware

Auch für die sehr verbreitete Virtualisierungs-Plattform VMware vSphere gibt es natürlich ein Inventory-Plugin. Bevor es zum Einsatz kommen kann, müssen Sie auf dem Control Host mindestens das Modul *pyvmomi* installieren:

```
# pip3 install pyvmomi
```

Sie wollen wahrscheinlich aber auch virtuelle Maschinen taggen und die Tags dann als Ansible-Gruppen abbilden lassen. Dazu müssen Sie *zusätzlich* noch das vSphere Automation SDK installieren:

```
# pip3 install --upgrade pip setuptools
# pip3 install git+https://github.com/vmware/vsphere-automation-sdk-python.git
```

Sie benötigen im zweiten Schritt tatsächlich ein sehr neues `pip`, sonst schlägt die Installation des SDKs gegebenenfalls fehl. Vielleicht wollen Sie das alles lieber in einer Virtualenv-Umgebung testen.

Wenn alle benötigten Python-Komponenten installiert sind, müssen Sie das Inventory-Plugin noch in der *ansible.cfg* aktivieren:

```
[inventory]
enable_plugins = vmware_vm_inventory
```

Listing 7.15 »ansible.cfg«: Ausschnitt

Wichtig

Beachten Sie dabei die `[inventory]`-Sektion, die es in Ihrer Konfiguration bislang wahrscheinlich noch nicht gibt!

Kommen wir nun zur eigentlichen Ansible-Integration. Da wir hier ein neues Inventory verwalten möchten, sollten wir in unserem Inventory-Ordner auch ein neues Unterverzeichnis anlegen, etwa *vmware_demo*:

```
$ cd inventories
$ mkdir vmware_demo && cd vmware_demo
```

In diesem Ordner erstellen Sie dann die benötigte YAML-Konfigurationsdatei. Das Plugin verlangt, dass der Dateiname auf *vmware.(yml|yaml) endet*, also nennen wir sie bspw. *demo.vmware.yml*. So sieht eine exemplarische Datei aus, die Sie bitte entsprechend anpassen:

```
---
plugin: vmware_vm_inventory
hostname: <VCENTER_SERVERNAME>
username: <USERNAME>
password: <PASSWORT>
validate_certs: False
with_tags: True
```

Listing 7.16 »inventories/vmware_demo/demo.vmware.yml«: Basis-Plugin-Konfiguration

Die Direktive `with_tags` können Sie aber nur dann verwenden, wenn Sie das vSphere Automation SDK installiert haben (siehe oben).

Nun sind Sie bereit für einen ersten Test. Mit folgendem Aufruf sehen Sie (hoffentlich) Ihr Inventar:

```
$ ansible-inventory -i demo.vmware.yml --list
```

Tags

Im vSphere Client können Sie Tags anlegen (unter Menü, Tags und benutzerdefinierte Attribute) und diese Tags dann virtuellen Maschinen zuweisen. Das Inventory-Plugin wird dann (mit der Einstellung `with_tags: True`) für jedes Tag eine Gruppe anlegen, die alle Hosts enthält, denen dieses Tag zugewiesen wurde.

Das ist auch sehr empfehlenswert, denn per Default gibt es nur relativ unschöne Gruppen wie `windows9Server64Guest` oder `other3xLinux64Guest`, die in der Regel nicht für eine weitergehende Parametrisierung (z. B. mittels `group_vars`) taugen.

7.5.6 Inventory-Plugins: Caching

Beim Arbeiten mit dynamischen Inventorys stellt sich mitunter schnell die Frage nach einem Cache. Wenn Sie bspw. das `aws_ecs`-Plugin verwenden und bei jedem Aufruf zehn bis 15 Sekunden warten müssen, kann das die Geduld schon auf eine harte

Probe stellen. Dankenswerterweise unterstützen viele Inventory-Plugins das Inventory-Caching; das `aws_ecs`-Plugin und das `vmware_vm_inventory` gehören dazu. Sehen Sie hier am Beispiel von `aws_ecs`, wie Sie in der Plugin-Konfiguration das Caching aktivieren:

```
---
[…]

cache: yes
cache_plugin: jsonfile
cache_timeout: 3600
cache_connection: ~/.ansible/inventory_cache
cache_prefix: aws_ec2
```

Listing 7.17 »demo.aws_ec2.yml«: Konfiguration mit Cache

Das Ganze erinnert sehr stark an den bereits bekannten Fakten-Cache. In der Tat kann das Caching alternativ auch in der Ansible-Konfiguration aktiviert werden:

```
[…]

[inventory]
cache = yes
cache_plugin = jsonfile
cache_timeout = 3600
cache_connection = ~/.ansible/inventory_cache
```

Listing 7.18 »ansible.cfg«: Ausschnitt: aktivieren des Inventory-Caches

7.6 Inventory-Verzeichnisse

Zum Ende des Kapitels soll nicht unerwähnt bleiben, dass es in Ansible auch die Möglichkeit gibt, anstelle einer einzigen Inventory-Quelle ein ganzes Verzeichnis als Inventory zu benutzen.

Dieses muss dann irgendwelche Inventory-Quellen enthalten (welche auch in weiteren Unterverzeichnissen verteilt sein können), und die Summe aller dieser Quellen bildet dann das Inventory.

Ich empfehle Ihnen aber, diese Methode lieber nicht einzusetzen (wenn es sich vermeiden lässt), denn selbst mit viel Disziplin kann man hier nur schwer den Überblick behalten (von den Problemen, die entstehen wenn widersprüchliche Angaben in verschiedenen Inventory-Quellen existieren, ganz zu schweigen).

Kapitel 9
Die Modul-Bibliothek

Ich gebe Ihnen in diesem Kapitel eine Kurzübersicht über die typischerweise am häufigsten eingesetzten Ansible-Module. Das kann und soll aber kein Ersatz für die offizielle Dokumentation sein, die Sie bitte stets zurate ziehen, wenn Sie detailliertere Informationen benötigen. Ein Beispiel: Sie wollen mit dem `user`-Modul arbeiten, um Linux-User anzulegen. Der schnellste Weg zur Online-Dokumentation führt heutzutage über eine Suchmaschine Ihrer Wahl; suchen Sie in diesem Fall einfach nach *»Ansible user module«*. Der erste Treffer sollte in der Regel der Link zur passenden Dokumentationsseite sein. Die Seite *http://docs.ansible.com/ansible/modules_by_category.html* bietet einen nach Kategorien geordneten Einstieg in die Modul-Dokumentation. (Falls Sie wissen möchten, was es alles so gibt.)

Sie können die Dokumentation zu einem Modul auch offline lesen mit:

```
$ ansible-doc <MODULNAME>
```

Eine Auflistung aller in Ihrer Ansible-Version verfügbaren Module bekommen Sie schließlich mit

```
$ ansible-doc -l
```

auf den Bildschirm. Ich gebe Ihnen in der folgenden Kurzübersicht zu jedem Modul ein oder zwei Aufrufbeispiele; ein möglichst kurzes und gegebenenfalls eines, das weitergehende Aufrufmöglichkeiten demonstriert.

9.1 Module zur Kommandoausführung

command – Kommando ausführen

Zur Kommandoausführung stehen die folgenden Module zur Verfügung:

```
- name: Dateien suchen und Ergebnis in Variable sichern
  command: find /etc -name *.conf
  register: find_cmd

- name: Ergebnis einer Sortierung in /tmp ablegen
  command:
    chdir: /tmp
```

```
    cmd: sort /etc/passwd -o sorted.txt
    creates: sorted.txt
```

Das zweite Beispiel zeigt den Wechsel des Arbeitsverzeichnisses mit `chdir` und einen »Idempotenzhinweis« mit `create`. Online-Dokumentation:
http://docs.ansible.com/ansible/modules/command_module.html

shell – Kommando über eine Shell ausführen

```
- name: Programme mit "a" am Anfang zählen
  shell: "ls -l /usr/bin/a* | wc -l"

- name: Brace-Expansion (benötigt eine Bash)
  shell:
    cmd: echo test{1,2,3}
    executable: /bin/bash
```

Dieses Modul ähnelt dem `command`-Modul, nur wird der Befehl hier per Default via `/bin/sh` ausgeführt. Wenn Sie Shell-Mechanismen benötigen, die von einer klassischen Bourne Shell nicht unterstützt werden, können Sie den gewünschten Shell-Interpreter auch angeben (siehe das zweite Beispiel). Online-Dokumentation:
http://docs.ansible.com/ansible/modules/shell_module.html

raw – Kommando ohne Python-Subsystem ausführen

```
- name: Python 3 auf einem Debian-Target Host installieren
  raw: apt-get -y install python3
```

Am häufigsten wird dieses Modul wahrscheinlich eingesetzt, um auf einem Zielsystem ohne Python überhaupt erst mal ein Python-Paket zu installieren. Aus technischer Sicht ist es mit dem `shell`-Modul vergleichbar; der `executable`-Parameter steht hier ebenfalls zur Verfügung. Online-Dokumentation:
http://docs.ansible.com/ansible/modules/raw_module.html

script – lokales Skript übertragen und ausführen

```
- name: Skript übertragen und ausführen
  script: /usr/local/bin/myscript.sh --opt1 1234
```

Der Name dieses Moduls ist etwas irreführend, da Sie jede Art von Programm verwenden können. Natürlich werden es in der Praxis meist Shell-, Perl- oder Python-Skripte sein. Interessant ist noch, dass (genau wie beim `raw`-Modul) kein Python auf der Zielmaschine vorausgesetzt wird. Ansonsten sind die Parametrisierungsmöglichkeiten vergleichbar mit denen des `shell`-Moduls. Online-Dokumentation:
http://docs.ansible.com/ansible/modules/script_module.html

9.2 Module zur Paketverwaltung

Die folgenden Module dienen zur Paketverwaltung.

apt – Paketmanagement auf Debian/Ubuntu-Systemen

```
- name: Das Pendant zu apt-get update
  apt:
    update_cache: yes
    cache_valid_time: 3600

- name: Paket "tmux" installieren
  apt:
    name: tmux
```

Zur Verwaltung von Repositorys und Schlüsseln stehen die Module `apt_repository` und `apt_key` zur Verfügung.

Online-Dokumentation: *http://docs.ansible.com/ansible/modules/apt_module.html*

yum, dnf – Paketmanagement auf CentOS-/Red-Hat-/Fedora-Systemen

```
- name: Paket "tmux" installieren
  yum:
    name: tmux

- name: Paket "tree" installieren
  dnf:
    name: tree
```

Ab CentOS 8 ist `dnf` der Default-Paketmanager, `yum`-Aufrufe sind aber weiterhin möglich. Aus Sicht von Ansible verhält es sich genauso; sofern Sie also keine speziellen `dnf`-Features benötigen, können Sie wie gewohnt beim `yum`-Modul bleiben.

Zur Verwaltung von Repositorys steht das Modul `yum_repository` zur Verfügung.

Online-Dokumentation: *http://docs.ansible.com/ansible/modules/yum_module.html*

zypper – Paketmanagement auf SUSE-Systemen

```
- name: Paket "tmux" installieren
  zypper:
    name: tmux
```

Zur Verwaltung von Repositorys steht das Modul `zypper_repository` zur Verfügung.

Online-Dokumentation:
http://docs.ansible.com/ansible/modules/zypper_module.html

package – generisches Paketmanagement

```
- name: Paket "tmux" installieren
  package:
    name: tmux
```

Das package-Modul ist der kleinste gemeinsame Nenner des Paketmanagements. Unter der Haube wird einfach das eigentliche Modul (apt, yum etc.) aufgerufen. Mehr dazu in der Online-Dokumentation:
http://docs.ansible.com/ansible/modules/package_module.html

package_facts – Paketinformationen als Fakten darstellen

```
- name: Fakten über installierte Pakete einsammeln
  package_facts:

- debug: msg="Apache ist installiert"
  when: ansible_facts['packages']['apache2'] is defined

- debug: var=ansible_facts['packages']['apache2']
```

Das Modul package_facts integriert Informationen über installierte Distributionspakete in die Fakten. Die Datenstruktur pro Paket sieht für obigen Fall z. B. so aus:

```
"ansible_facts['packages']['apache2']": [
        {
            "arch": "amd64",
            "category": "httpd",
            "name": "apache2",
            "origin": "Debian",
            "source": "apt",
            "version": "2.4.38-3+deb10u3"
        }
    ]
```

Einige weitere (nicht näher beschriebene) Module

- cpanm – Perl-Module verwalten
- pip – Python-Module verwalten
- pear – PHP-Pakete verwalten

9.3 Module zur Verwaltung von Dateien und Dateiinhalten

Mit den nachstehenden Modulen können Sie sowohl Dateien, als auch Dateiinhalte verwalten.

copy – Dateien hochladen

```
- name: Beispieldatei "test1.txt" zum Zielhost kopieren
  copy:
    src: test1.txt
    dest: /tmp

- name: Dabei auch Zugriffs- und Eigentumsrechte verwalten
  copy:
    src: test1.txt
    dest: /tmp
    owner: user1
    group: users
    mode: 0600
```

Wenn die Quelldatei durch einen relativen Pfad spezifiziert wird, sucht Ansible in einem Ordner *files/* parallel zum Playbook und danach direkt parallel zum Playbook. Bei Verwendung in einer Rolle wird zunächst im Rollen-Ordner *files/* gesucht.

Der mode kann numerisch oder symbolisch (z. B. 'u+rw') angegeben werden. Außerdem steht das Schlüsselwort preserve zur Verfügung, mit dem die Zieldatei dieselben Rechte wie die Quelldatei bekommt.

Achtung

Es gibt zwei funktionierende Möglichkeiten, einen numerischen Mode anzugeben:

1. mithilfe einer Zahl mit führender 0 (wie im obigen Beispiel), sodass der YAML-Parser von Ansible sofort eine Oktalzahl erkennt
2. mit einem *quotierten* String wie '644' oder '1777', sodass Ansible intern selbst konvertieren kann

Wenn Sie eine Zahl auf andere Art spezifizieren, wird dies nicht zum gewünschten Ergebnis führen!

Online-Dokumentation: *http://docs.ansible.com/ansible/modules/copy_module.html*

template – Dateien mit Jinja verarbeiten und hochladen

```
- name: Template füllen und hochladen
  template:
    src: index.html.j2
    dest: /srv/www/htdocs/index.html
```

Dieses Modul ist in der Verwendung nahezu indentisch mit dem copy-Modul, nur dass der Input zusätzlich mit der Jinja-Engine verarbeitet wird. (Falls Ihnen der Begriff

»Jinja« nichts sagt, sollten Sie in jedem Fall Abschnitt 6.5 lesen.) Beachten Sie bitte, dass bei relativen Pfadangaben hier zusätzlich im Ordner *templates/* nach der Quelldatei gesucht wird (und *nicht* in *files/*!).

Online-Dokumentation:
http://docs.ansible.com/ansible/modules/template_module.html

file – Dateien und Dateiattribute verwalten

```
- name: Zugriffs- und Eigentumsrechte setzen
  file:
    path: /etc/ssl/private
    owner: root
    group: ssl-cert
    mode: 0710
```

Mit `file` können Sie auch Dateien und Verzeichnisse anlegen oder löschen, Soft- und Hardlinks verwalten und einiges mehr.

Gerade das Anlegen von Links geht mit diesem Modul aber nicht immer völlig leicht von der Hand, deswegen dazu noch einige Bemerkungen. Angenommen, Sie möchten mit Ansible erreichen, was Sie in der Shell wie folgt lösen:

```
$ ln -s <ALTER_NAME> <NEUER_NAME>
```

So merke ich mir seit ewigen Zeiten die Reihenfolge beim `ln`-Kommando: Erst kommt der Pfad zur alten (meist schon existenten) Datei, dann kommt der Linkname – also ein neuer Name für diese Datei. Der entsprechende Ansible-Task sieht dann so aus:

```
- name: Symbolischen Link setzen
  file:
    src:  ALTER_NAME
    dest: NEUER_NAME
    state: link
```

Einen Hardlink bekämen Sie mit `state: hard`. Die Syntax des `file`-Moduls kommt Ihnen also hier entgegen, wenn Sie sich die Funktion des `ln`-Kommandos sowieso mit »Quelle« und »Ziel« merken. Problematisch wird es nun noch in zwei Situationen:

1. Die alte Datei bzw. die Quelle existiert nicht (das wäre ja für das `ln`-Kommando bei Symlinks kein Problem).
2. Der neue Name, also der Link-Pfad, existiert bereits, ist aber kein Symlink (sondern ein Verzeichnis, eine normale Datei oder was auch immer).

Diese Fälle wären für Ansible jeweils ein Fehler, den Sie aber mit dem zusätzlichen Modul-Parameter **`force: yes`** vermeiden können.

Online-Dokumentation: *http://docs.ansible.com/ansible/modules/file_module.html*

stat – Informationen über Dateien gewinnen

```
- name: Dateieigenschaften von /etc/ssl bestimmen
  stat:
    path: /etc/ssl
  register: p

- debug:
    msg: /etc/ssl ist ein existierendes Verzeichnis
  when: p.stat.exists and p.stat.isdir
```

Ein Tipp: Schauen Sie sich einmal die komplette Struktur der registrierten Variablen an (`debug: var=p`). Da ist noch einiges mehr an Informationen enthalten.

Online-Dokumentation: *http://docs.ansible.com/ansible/modules/stat_module.html*

lineinfile – Zeilen in Textdateien verwalten

```
- name: Zeile ändern
  lineinfile:
    path: /etc/selinux/config
    regexp: '^SELINUX='
    line: 'SELINUX=disabled'

- name: Zeile an festgelegter Stelle einfügen
  lineinfile:
    path: /etc/ssh/sshd_config
    insertafter: '^#ListenAddress'
    regexp: '^ListenAddress'
    line: 'ListenAddress 192.168.150.20'
```

Für einen erfolgreichen Einsatz dieses Moduls (sowie der Module `blockinfile` und `replace`) müssen Sie einigermaßen sattelfest im Umgang mit regulären Ausdrücken sein. Wenn Sie hier Nachholbedarf haben oder überhaupt einen Einstieg brauchen, dann schauen Sie doch mal in Anhang B. Für die absoluten Feinheiten des konkreten »Dialekts« gehen Sie bitte auf *http://docs.python.org/3/library/re.html*.

Online-Dokumentation:
http://docs.ansible.com/ansible/modules/lineinfile_module.html

blockinfile – Textpassagen in Dateien verwalten

```
- name: Konfiguration für virtuelles Interface eintragen
  blockinfile:
    path: /etc/network/interfaces
    block: |
      iface ens33:0 inet static
```

```
        address 10.0.0.1/24
        netmask 255.255.255.0
```

Das Modul verwaltet Textpassagen, die von konfigurierbaren Markierungzeilen umgeben sind. Zur Positionierung stehen mit `lineinfile` vergleichbare Möglichkeiten zur Verfügung. Online-Dokumentation:
http://docs.ansible.com/ansible/modules/blockinfile_module.html

replace – suchen und ersetzen in Textdateien

```
- name: PasswordAuthentication ggf. einkommentieren und auf "no" setzen
  replace:
    path: /etc/ssh/sshd_config
    regexp: '^#?(PasswordAuthentication).*'
    replace: '\1 no'
```

Das Modul ersetzt *alle* Vorkommen von `regexp` in der Datei durch den `replace`-Ausdruck. Falls `replace` fehlt, werden die Vorkommen gelöscht. Online-Dokumentation: *http://docs.ansible.com/ansible/modules/replace_module.html*

unarchive – Archive hochladen und auspacken

```
- name: Archiv-Datei in /srv/www/webapps auspacken
  unarchive:
    src: 'roundcubemail-1.3.4-complete.tar.gz'
    dest: /srv/www/webapps
    owner: www-data
    group: www-data
    creates: '/srv/www/webapps/roundcubemail-1.3.4'
```

Dieses Modul kann mit den Archivformaten *.zip*, *.tar*, *.tar.gz*, *.tar.bz2* und *.tar.xz* umgehen. Wenn die Quelle als relativer Pfad spezifiziert wird, ist die Suchstrategie dieselbe wie beim `copy`-Modul. Die Zielverzeichnisse müssen existieren. Online-Dokumentation: *http://docs.ansible.com/ansible/modules/unarchive_module.html*

Einige weitere (nicht näher beschriebene) Module

- `acl` – File-ACLs verwalten
- `fetch` – wie `copy`, nur anders herum
- `find` – das `find`-Kommando als Modul
- `ini_file` – Inhalte von INI-Dateien verwalten
- `rsync` – das `rsync`-Kommando als Modul
- `tempfile` – temporäre Dateien oder Verzeichnisse erzeugen
- `xml` – Inhalte von XML-Dateien verwalten

9.4 Module für weitere typische Verwaltungsaufgaben

Für weitere typische Verwaltungsaufgaben stehen Ihnen u. a. folgende Module zur Verfügung:

service – Dienste starten, stoppen, neu starten, ...

```
- name: Apache starten und in Autostart integrieren
  service:
    name: httpd
    state: started
    enabled: yes
```

Online-Dokumentation:
http://docs.ansible.com/ansible/modules/service_module.html

service_facts – Service-Informationen als Fakten darstellen

```
- name: Fakten über Dienste einsammeln
  service_facts:

- debug: msg="Apache läuft"
  when: ansible_facts['services']['apache2']['state'] == 'running'

- debug: var=ansible_facts['services']['apache2']
```

Das Modul `service_facts` integriert Informationen über installierte Distributionspakete in die Fakten. Die Datenstruktur pro Dienst sieht für den obigen Fall z. B. so aus:

```
"ansible_facts['services']['apache2']": {
        "name": "apache2",
        "source": "sysv",
        "state": "running"
    }
```

Online-Dokumentation:
http://docs.ansible.com/ansible/modules/service_facts_module.html

cron – Cronjobs verwalten

```
- name: Cronjob unter /etc/cron.d/ einrichten
  cron:
    cron_file: certbot
    user: root    name: Run "certbot renew" once per week
    special_time: weekly
    job: certbot renew --post-hook "systemctl restart postfix apache2"
```

Natürlich stehen zur Zeitspezifikation auch Parameter wie `minute`, `hour`, `day` … zur Verfügung.

Online-Dokumentation:
http://docs.ansible.com/ansible/modules/cron_module.html

hostname – den Hostnamen ändern

```
- name: Hostnamen auf "web01" setzen
  hostname:
    name: web01
```

Online-Dokumentation:
http://docs.ansible.com/ansible/modules/hostname_module.html

user, group – Benutzer- und Gruppenverwaltung

```
- name: Gruppe "vmail" anlegen
  group:
    name: vmail
    gid: 5000

- name: User "vmail" anlegen
  user:
    name: vmail
    uid: 5000
    group: vmail
    home: /srv/imap
    shell: /bin/false
```

Online-Dokumentation:

- *http://docs.ansible.com/ansible/modules/user_module.html*
- *http://docs.ansible.com/ansible/modules/group_module.html*

firewalld – Firewalld-basierte Firewalls verwalten

```
- name: HTTP-Port 80/tcp öffnen
  firewalld:
    service: http
    state: enabled
    permanent: yes
    immediate: yes
```

Dieses Modul benötigt auf den Zielsystemen entsprechende Python-Bindings zum Firewalld-Management. In seltenen Fällen müssten Sie also vorher das Paket *python3-firewall* nachinstallieren. Ansonsten muss man an dieser Stelle sagen: Ohne

fundiertes Wissen zu Firewalld wird man auch mit diesem Modul nicht besonders weit kommen!

Online-Dokumentation:
http://docs.ansible.com/ansible/modules/firewalld_module.html

reboot – Maschinen rebooten

```
- name: Maschine neu starten
  reboot:
```

Online-Dokumentation:
http://docs.ansible.com/ansible/modules/reboot_module.html

Einige weitere (nicht näher beschriebene) Module

- `authorized_key` – SSH-Keys verwalten
- `filesystem` – Dateisysteme erzeugen
- `getent` – Wrapper um das Linux-`getent`-Kommando
- `locale_gen` – Locales verwalten
- `lvg`, `lvol` – LVM-Devices und -Volumegruppen bzw. Logical Volumes verwalten
- `mount` – Mounts verwalten

9.5 Module zur Interaktion mit Netzwerk-Services

Zur Interaktion mit Netzwerk-Services sind diese Module nützlich.

get_url – Download von Ressourcen

```
- name: Pinguin-Bild herunterladen
  get_url:
    url: https://upload.wikimedia.org/wikipedia/commons/a/af/Tux.png
    dest: /tmp/tux.png
```

Mit diesem Modul können Sie eine HTTP-, HTTPS- oder FTP-Ressource auf ein Zielsystem herunterladen. Ohne weitere Einstellungen braucht das System dazu direkten Zugang zu dieser Ressource; die Nutzung von Proxys ist aber möglich durch die Umgebungsvariablen `<protokoll>_proxy`. Siehe dazu gegebenenfalls Abschnitt 6.12. Online-Dokumentation: *http://docs.ansible.com/ansible/modules/get_url_module.html*

uri – Kommunikation mit Webservices

```
- name: Einen Chuck Norris-Witz besorgen
  uri:
```

```
    url: http://api.icndb.com/jokes/random
    return_content: yes
  register: this

- debug: var=this.json.value.joke
```

Neben solchen simplen Abfragen hat das Modul `uri` eine große Bandbreite an Möglichkeiten: POST-Requests, Authentifizierung, Cookies usw.

Online-Dokumentation: *http://docs.ansible.com/ansible/modules/uri_module.html*

git – Interaktion mit Git-Servern

```
- name: Beispielrepository klonen
  git:
    repo: https://github.com/ansible-buch/docker-installer.git
    dest: /tmp/ansible-buch/docker-installer
```

Voraussetzung für den Betrieb ist das Vorhandensein der Git-Software auf den entsprechenden Systemen. Sie müssen also gegebenenfalls in einem vorangehenden Task erst einmal das Paket *git* installieren.

Ansonsten bietet auch das `git`-Modul noch viel mehr Möglichkeiten, wie z. B. SSH-Key-Authentifizierung oder Checkout von Branches (Parameter `version`).

Online-Dokumentation: *http://docs.ansible.com/ansible/modules/git_module.html*

9.6 Spezialmodule (Kontrollflusssteuerung etc.)

In diesem Abschnitt finden Sie eine Auswahl von Modulen, mit denen Sie sehr spezielle (meist interne) Aktionen auslösen können.

meta – Verschiedene Ansible-Aktionen auslösen

```
- name: Wir beenden das jetzt
  meta: end_play
```

Die Menge der verfügbaren Metaaktionen ist im Lauf der Jahre immer weiter angestiegen. Ansible ≥ 2.8 kennt acht Aktionen:

- `clear_facts`
  Fakten und Fakten-Cache löschen
- `clear_host_errors`
  Sehr missverständlich – nimmt lediglich Unreachable Hosts wieder mit an Bord. Und auch das nur, wenn überhaupt noch ein Host übrig geblieben ist.

- `end_host`
  Beendet das Play für den aktuellen Host (ohne Fehler).
- `end_play`
  Beendet das Play für alle Hosts (ohne Fehler).
- `flush_handlers`
  »Vorzeitiger« Aufruf von Handlern; siehe Abschnitt 5.10.2.
- `noop`
  Tut nichts (das muss auch mal sein).
- `refresh_inventory`
  Löst einen Reload des Inventorys aus (ergibt nur Sinn bei dynamischen Inventorys).
- `reset_connection`
  Wirft die persistenten SSH-Verbindungen weg, sodass eventuelle Folge-Tasks dann neue Verbindungen aufbauen würden.

Online-Dokumentation:
http://docs.ansible.com/ansible/modules/meta_module.html

debug – beim Playbook-Lauf Ausgaben erzeugen

```
- debug: msg="Hallo"

- debug: var=ansible_default_ipv4
```

Auf die Benennung der Tasks mittels `name` wird sehr oft verzichtet, schon allein, weil ein `debug`-Task oft nicht dauerhaft im Play verbleiben soll. Der Parameter `var` ist dann praktisch, wenn Sie möglichst einfach den Inhalt einer Variablen sichtbar machen möchten. Online-Dokumentation:
http://docs.ansible.com/ansible/modules/debug_module.html

fail – Fehlschlag auslösen

```
- set_fact:
    hour: "{{ lookup('pipe', 'date +%H') }}"

- fail:
    msg: "Es ist heute einfach schon zu spät."
  when: hour|int > 20
```

Mit dem `fail`-Modul können Sie gezielt einen Fehlschlag auslösen und das Play damit für die jeweiligen Hosts beenden. In alten Ansible-Tagen, bevor `meta: end_play` erfunden war, war dies die einzige Möglichkeit, aus einem Play auszusteigen. Online-Dokumentation: *http://docs.ansible.com/ansible/modules/fail_module.html*

pause – pausieren und optional Eingaben einlesen

```
- pause:
    prompt: Bitte machen Sie nun das Licht im Serverraum aus

- name: Der CPU Zeit zum Abkühlen geben
  pause:
    seconds: 30

- pause:
    prompt: Geben Sie bitte noch Ihren Namen ein
  register: this

- debug: msg="Auf Wiedersehen, {{this.user_input}}!"
```

Mit dem `pause`-Modul können Sie die Ausführung eines Playbooks anhalten bzw. für eine gewisse Zeit pausieren.

[!]

Achtung

Bitte wenden Sie das Warten auf interaktive Eingaben *nicht* an, wenn Ihr Playbook möglicherweise einmal in nichtinteraktiven Umgebungen wie Cron oder Ansible AWX/Tower ablaufen soll!

Online-Dokumentation:
http://docs.ansible.com/ansible/modules/pause_module.html

wait_for – auf gewisse Ereignisse warten

```
- name: Warten, bis die Datei /tmp/foo auftaucht
  wait_for:
    path: /tmp/foo

- name: Warte max. 300 Sekunden, bis Port 8000 offen ist
  wait_for:
    port: 8000
```

Mit dem `wait_for`-Modul können Sie im Play warten, bis eine gewisse Bedingung eintritt. Die maximale Wartezeit beträgt per Default 300 Sekunden (Parameter `timeout`). Neben den gezeigten Möglichkeiten können Sie auch auf das Schließen eines Ports warten, oder Sie können Dateiinhalte oder Sockets mit einem regulären Ausdruck vergleichen und dabei warten, bis das Muster passt.

Online-Dokumentation:
http://docs.ansible.com/ansible/modules/wait_for_module.html

wait_for_connection – warten, bis ein Target Host erreichbar ist

```
- name: Warte max. 600 Sekunden, bis der Target Host erreichbar ist
  wait_for_connection:
    timeout: 600
```

Das Timeout von 600 Sekunden ist aber ohnehin der Default und müsste daher nicht eigens angegeben werden.

Online-Dokumentation:
http://docs.ansible.com/ansible/modules/wait_for_connection_module.html

assert – sicherstellen, dass gewisse Bedingungen erfüllt sind

```
- name: Es muss ein Debian 9 oder 10 sein
  assert:
    that:
      - ansible_distribution == "Debian"
      - ansible_distribution_major_version is version('9', '>=')
      - ansible_distribution_major_version is version('10', '<=')
    fail_msg: "Wir unterstützen leider nur Debian in Version 9 oder 10"
```

Das `assert`-Modul stellt sicher, dass alle angegebenen Bedingungen zutreffen, ansonsten bricht die Verarbeitung mit einem Fehler ab. Wir hatten bereits in Abschnitt 6.1.10 ausführlicher darüber gesprochen.

Online-Dokumentation:
http://docs.ansible.com/ansible/modules/assert_module.html

set_fact – während der Laufzeit Variablen setzen

```
- name: Zwei Variablen setzen
  set_fact:
    foo: 100
    bar: 200

- debug: msg="foo = {{foo}}, bar = {{bar}}"
```

Mit `set_fact` können Sie während der Laufzeit host-spezifische Variablen setzen. Wir hatten bereits in Abschnitt 6.1.4 ausführlicher darüber gesprochen.

Online-Dokumentation:
http://docs.ansible.com/ansible/modules/set_fact_module.html

Auf einen Blick

1 Einführung und Installation 23
2 Basiseinrichtung und erstes Inventory-Management 41
3 Ad-hoc-Kommandos und Patterns 53
4 Die Konfigurations- und Serialisierungssprache YAML 63
5 Playbooks und Tasks: die Grundlagen 71
6 Playbooks und Tasks: fortgeschrittene Methoden 97
7 Inventory-Management: fortgeschrittene Methoden 147
8 Modularisierung mit Rollen und Includes 167
9 Die Modul-Bibliothek 185
10 Webinterfaces 201
11 Weitere Tools und Techniken 221
12 Ansible als Orchestrierungswerkzeug 251
13 Ansible und Docker 277
14 Windows-Hosts verwalten 303
15 Kochrezepte, Howtos und Best Practices 313
16 Eigene Module erstellen 331
17 Callback-Plugins 345
18 Was könnte noch besser sein, bzw. was fehlt noch? 357

Inhalt

Materialien zum Buch 15
Vorwort 17
Über dieses Buch 19

1 Einführung und Installation 23

1.1 Was ist Ansible? 23
1.2 Was ist Ansible nicht? 25
1.3 Setup/Laborumgebung 26
1.3.1 Zugangsdaten für die Systeme der Laborumgebung 30
1.3.2 Falls Probleme auftreten 30
1.4 Ansible-Installation auf dem Control Host 31
1.5 Authentifizierung und Autorisierung auf den Target Hosts 34
1.6 Einrichten der SSH-Public-Key-Authentifizierung 36
1.7 Ein Ad-hoc-Test ohne jegliche Konfiguration 37
1.8 Noch ein Hinweis zur Migration von älteren Versionen 39

2 Basiseinrichtung und erstes Inventory-Management 41

2.1 Verzeichnisstruktur einrichten 41
2.2 Grundkonfiguration (»ansible.cfg«) 42
2.3 Erstellen und Verwalten eines statischen Inventorys 43
2.4 Inventory-Aliase 46
2.5 Jenseits von Ping 46
2.6 Alternative Inventorys 50

3 Ad-hoc-Kommandos und Patterns 53

3.1 Ad-hoc-Kommandos ... 53
3.2 Use cases ... 55
3.3 Idempotenz ... 56
3.4 Interne Funktionsweise ... 57
3.4.1 Parallele Ausführung ... 57
3.4.2 Persistente Verbindungen ... 58
3.4.3 Was passiert beim Aufruf eines Moduls? ... 60
3.5 Die Ansible-Konsole ... 61
3.6 Patterns zum Adressieren von Hosts ... 62

4 Die Konfigurations- und Serialisierungssprache YAML 63

4.1 Syntax und Struktur ... 63
4.2 YAML-Files editieren ... 63
4.3 Listen und Maps ... 66
4.4 Verschachtelte Strukturen ... 66
4.5 Block-Ausdrücke ... 68
4.6 Das Nichts in YAML ... 69
4.7 Anchors und References ... 69

5 Playbooks und Tasks: die Grundlagen 71

5.1 Hallo Ansible – das allererste Playbook ... 71
5.2 Formulierung von Tasks ... 74
5.3 Beenden von Plays ... 76
5.4 Der problematische Doppelpunkt ... 77
5.5 Kommandoaufrufe mit den Modulen »command« und »shell« ... 78
5.6 Fehler und Retry-Files ... 81
5.7 Tags ... 84

5.8 Das Kommando »ansible-playbook« ... 85
5.9 Eine exemplarische Apache-Installation ... 86
5.9.1 Schritt für Schritt ... 87
5.9.2 Das komplette Playbook ... 88
5.9.3 »--start-at-task«, »--check«, »--diff« ... 89
5.10 Handler: Tasks nur bei Changes durchführen ... 91
5.10.1 Schritt für Schritt ... 91
5.10.2 Handler ... 92
5.10.3 Das komplette Playbook bis hierhin ... 95

6 Playbooks und Tasks: fortgeschrittene Methoden 97

6.1 Variablen ... 97
6.1.1 Play Vars ... 97
6.1.2 Extra Vars ... 98
6.1.3 Präzedenzen ... 98
6.1.4 »set_fact« ... 99
6.1.5 »group_vars« ... 100
6.1.6 »host_vars« ... 102
6.1.7 »vars_files«: Variablen in beliebigen externen Dateien ... 103
6.1.8 Prompting ... 103
6.1.9 Zugriffe auf komplexe Strukturen ... 104
6.1.10 Seid ihr alle da? ... 105
6.2 Facts und implizite Variablen ... 106
6.2.1 Facts ... 106
6.2.2 Cachen von Facts ... 108
6.2.3 Implizite Variablen ... 109
6.2.4 Ein Beispiel ... 109
6.2.5 Externe Informationsbeschaffer: »facter« und »ohai« ... 111
6.2.6 Noch nicht genug Fakten? »/etc/ansible/facts.d«! ... 111
6.3 Bedingte Ausführung mit »when« ... 111
6.4 Systemunterschiede ausgleichen – wie denn jetzt? ... 113
6.4.1 Die plumpe Methode ... 113
6.4.2 Die solide Methode ... 114
6.4.3 Die trickreiche Methode ... 116
6.4.4 Die modulare Methode ... 116
6.4.5 Das komplette Playbook bis hierhin ... 118

6.5 Jinja und Templates ... 120
6.5.1 Begriffsklärung: Templates und Template-Engines ... 120
6.5.2 Eine individuelle Startseite für unsere Apache-Server ... 121
6.5.3 Schnelles Testen von Jinja-Templates ... 123
6.5.4 Jinja-Syntax: Ausgabeausdrücke, Anweisungen, Kommentare ... 125
6.5.5 Filter ... 127
6.6 Schleifen ... 127
6.6.1 Iteration über eine Liste mit »with_items« oder »with_list« ... 128
6.6.2 Iteration über eine Map mit »with_dict« ... 130
6.6.3 Iteration über eine generierte Folge mit »with_sequence« ... 131
6.6.4 Verschachtelte Schleife mit »with_subelements« ... 132
6.6.5 Tasks wiederholen mit »until« ... 133
6.6.6 Mehr Kontrolle mit »loop_control« ... 134
6.7 Fehlerbehandlung mit »failed_when« und »ignore_errors« ... 137
6.8 Blocks (und noch mal Fehlerbehandlung) ... 138
6.9 Asynchrone Ausführung ... 140
6.10 Lokale Tasks ... 141
6.11 Lookup-Plugins ... 143
6.12 Umgebungsvariablen setzen ... 145

7 Inventory-Management: fortgeschrittene Methoden 147

7.1 Das Kommando »ansible-inventory« ... 147
7.2 Verschachtelte Gruppen ... 148
7.3 »On-the-fly«-Inventorys erstellen mit »add_host« ... 149
7.4 Dynamische Gruppen mit »group_by« ... 151
7.5 Dynamische bzw. externe Inventorys ... 153
7.5.1 Beispiel: ein Inventory-Skript in Perl ... 154
7.5.2 Verwenden von Inventory-Plugins ... 158
7.5.3 AWS EC2 ... 158
7.5.4 Hetzner Cloud ... 161
7.5.5 VMware ... 163
7.5.6 Inventory-Plugins: Caching ... 164
7.6 Inventory-Verzeichnisse ... 165

8 Modularisierung mit Rollen und Includes 167

8.1 Erstellung und Verwendung von Rollen ... 167
8.1.1 Das Rollenkonzept in Ansible ... 167
8.1.2 Ein einfaches Beispiel einer Rolle ... 168
8.1.3 Rollen verwenden ... 169
8.1.4 Abhängigkeiten zwischen Rollen ... 171
8.1.5 Erstellen neuer Rollen ... 172
8.2 Ansible Galaxy ... 173
8.3 Verwendung von Imports/Includes ... 174
8.4 Noch mal Apache ... 175
8.5 Dokumentation ... 179
8.5.1 Code-Konventionen ... 180
8.5.2 »README.md« ... 181

9 Die Modul-Bibliothek 185

9.1 Module zur Kommandoausführung ... 185
9.2 Module zur Paketverwaltung ... 187
9.3 Module zur Verwaltung von Dateien und Dateiinhalten ... 188
9.4 Module für weitere typische Verwaltungsaufgaben ... 193
9.5 Module zur Interaktion mit Netzwerk-Services ... 195
9.6 Spezialmodule (Kontrollflusssteuerung etc.) ... 196

10 Webinterfaces 201

10.1 Vorbereitungen zum Betrieb ... 201
10.2 Ansible Configuration Management Database (ansible-cmdb) ... 203
10.3 Voraussetzungen für anspruchsvollere Anwendungen ... 205
10.3.1 Der Reverse-Proxy Traefik ... 206
10.3.2 Der Git-Server Gitea ... 207
10.3.3 Einchecken unseres Projekts ... 208

10.4 Ansible Tower/AWX ... 210
10.4.1 Test-Setup und erste Anmeldung ... 211
10.4.2 Exemplarische Verwendung ... 212
10.5 Polemarch ... 213
10.5.1 Test-Setup und erste Anmeldung ... 213
10.5.2 Exemplarische Verwendung ... 214
10.5.3 (Vorsichtiges) Fazit ... 215
10.6 Jenkins ... 215
10.6.1 Test-Setup ... 215
10.6.2 Exemplarische Verwendung ... 216
10.6.3 Fazit ... 217
10.7 Weitere, hier nicht näher betrachtete Möglichkeiten ... 218
10.8 Laborumgebung: nicht mehr benötigte Anwendungen beenden ... 218

11 Weitere Tools und Techniken 221

11.1 Die Ansible-Vault ... 221
11.1.1 Vor aller Technik ... 221
11.1.2 Erste Schritte ... 223
11.1.3 Mehrere Vault-Passwörter und weitere Vault-Kommandos ... 225
11.1.4 Ein Trick zum Wiederfinden von Variablen ... 225
11.1.5 Mehr Bequemlichkeit bzw. Automatisierbarkeit ... 226
11.1.6 Bequem und (möglichst) sicher mit GPG + pass ... 227
11.2 Debugging und Troubleshooting ... 230
11.2.1 Debug-Mode und Verbosity Level ... 230
11.2.2 Lesbarkeit von Ausgaben verbessern ... 231
11.2.3 Gathering Facts dauert zu lange ... 233
11.2.4 Der Playbook-Debugger ... 235
11.2.5 Statische Code-Analyse mit »ansible-lint« ... 237
11.2.6 Überwachen von Dateiänderungen mit »--check« und »--diff« ... 240
11.2.7 Last, but not least: das »debug«-Modul ... 242
11.3 Untersuchen von Konfigurationseinstellungen ... 244
11.4 Playbooks beschleunigen mit Pipelining ... 244
11.5 Die sprechende Kuh ... 245

11.6 Ansible im Pull-Mode ... 246
11.6.1 »ansible-pull«: Technik und Voraussetzungen ... 247
11.6.2 Erste Schritte ... 248
11.6.3 Die ganze Lösung ... 249
11.6.4 Was fehlt noch? ... 250

12 Ansible als Orchestrierungswerkzeug 251

12.1 Administrierst du noch oder orchestrierst du schon? ... 251
12.2 Ausflug in die Cloud: Vorbereitungen ... 252
12.3 Erste Server in der Cloud ... 254
12.4 Die Abarbeitungsreihenfolge beeinflussen ... 256
12.4.1 »serial« ... 257
12.4.2 Fehlerhafte Hosts im »serial«-Betrieb ... 258
12.4.3 Strategy-Plugins ... 259
12.5 Delegierung ... 264
12.6 Ein komplexeres Beispiel ... 267
12.7 Löschen von Workern ... 274
12.8 Rolling-Updates ... 275

13 Ansible und Docker 277

13.1 Installation von Docker ... 277
13.2 Docker-Module ... 279
13.2.1 Vorbereitungen und Vorüberlegungen ... 279
13.2.2 Überblick ... 281
13.3 Eine Beispielanwendung ... 286
13.4 Ansible und Docker Compose ... 290
13.5 Erstellen von Images ... 295
13.5.1 Erstellen von Images mit »docker build« ... 295
13.5.2 »ansible-bender« ... 296
13.5.3 Erstellen von Images mit »ansible-bender« ... 298
13.5.4 Erklärungen und Fazit ... 301

14 Windows-Hosts verwalten 303

14.1 WinRM ... 303
14.2 Vorbereitungen auf dem Control Host ... 304
14.3 Voraussetzungen auf der Windows-Seite und WinRM-Setup ... 305
14.4 Setup mit Active Directory/Kerberos ... 306
14.5 WinRM-Troubleshooting ... 308
14.6 Windows-Module ... 309

15 Kochrezepte, Howtos und Best Practices 313

15.1 Eine empfehlenswerte »ansible.cfg« ... 313
15.2 Suche in einer Liste von Maps ... 314
15.3 Erweiterung von Maps oder Listen während der Laufzeit ... 315
15.4 Zufallspasswörter generieren ... 316
15.5 Einfache Installer bauen ... 318
15.6 IP-Adresse eines Target Hosts bestimmen ... 320
15.7 firewalld managen (falls vorhanden) ... 323
15.8 Linux-Software-Updates einspielen ... 324
15.9 Ansible über einen Gateway- bzw. Jumphost ... 327
15.10 Host-spezifische Ressourcen verwalten ... 328

16 Eigene Module erstellen 331

16.1 Erste Schritte ... 331
16.2 Modul-Parameter ... 334
16.3 Module in Python ... 337
16.3.1 Ein Beispiel in Python ... 338
16.3.2 Erklärungen und weitere Möglichkeiten ... 340
16.3.3 Eingebettete Dokumentation ... 343

17 Callback-Plugins 345

17.1 Stdout-Plugins ... 345
17.1.1 »default« ... 346
17.1.2 »yaml« ... 347
17.1.3 »json« ... 348
17.1.4 »unixy« ... 348
17.1.5 »dense« ... 349
17.1.6 »minimal« ... 349
17.1.7 »oneline« ... 350
17.1.8 »debug« ... 350
17.1.9 »selective« ... 350
17.1.10 »counter_enabled« ... 351
17.2 Sonstige Plugins ... 351
17.3 Entwickeln eigener Callback-Plugins ... 353

18 Was könnte noch besser sein, bzw. was fehlt noch? 357

18.1 Lang laufende Tasks verfolgen ... 357
18.2 Abarbeitung einer Rolle beenden ... 358
18.3 Schleifen über Blöcke ... 360
18.4 Locking bei konkurrierenden Playbook-Aufrufen ... 361
18.5 Fazit ... 363

Anhang 365

A SSH (Secure Shell) ... 367
A.1 Voraussetzungen zur Nutzung von SSH ... 367
A.2 SSH-Client-Programme ... 368
A.3 Public-Key-Authentifizierung ... 369
A.4 SSH-Agenten ... 372

A.5 Fortgeschrittene Konfiguration und Nutzung 373
A.6 Windows als SSH-Client 379
A.7 pssh bzw. parallel-ssh 380

B Reguläre Ausdrücke 383
B.1 Motivation 383
B.2 Dialekte 384
B.3 Reguläre Ausdrücke – Basics 384
B.4 Reguläre Ausdrücke – erweiterte Möglichkeiten (PCRE) 386
B.5 Reguläre Ausdrücke – erweiterte Möglichkeiten (POSIX Extended RE) 388

Index 391