

# Ansible Workshop für Einsteiger

Veranstaltung:  
**Secure Linux Administration Conference 2019**

Author: **René Koch** <rkoch@rk-it.at>  
Stand: **21.05.2019**

Version: **1.2**

# 1 Inhaltsverzeichnis

1	Inhaltsverzeichnis.....	2
2	Was ist Ansible.....	4
2.1	Ansible Engine vs. Ansible Tower.....	4
3	Testumgebung.....	6
3.1	SSH.....	6
3.2	Übung.....	6
4	Installation von Ansible.....	7
4.1	Installation via Paket-Management-Tool.....	7
4.1.1	RPM.....	7
4.1.2	DEB.....	7
4.1.3	PIP.....	7
4.2	Übung.....	8
4.3	Ansible Test-VM einbinden.....	9
4.4	Verbindung testen.....	9
4.5	Übung.....	9
5	Ad-Hoc Commands.....	10
5.1	Einfache Kommandos.....	10
5.1.1	Auslesen der Uptime.....	10
5.1.2	Log-Dateien betrachten.....	10
5.1.3	Paket installieren.....	10
5.2	Übung.....	12
5.2.1	Lösung.....	13
6	Playbooks.....	14
6.1	Einfaches Playbook.....	15
6.1.1	Anlegen eines Playbooks zum Installieren von Apache.....	15
6.1.2	Files deployen.....	17
6.1.3	Templates.....	18
6.1.4	Handler.....	21
6.2	YAML-Syntax.....	22
6.2.1	Playbook Header.....	22
6.2.2	Kommentare.....	22
6.2.3	String.....	22
6.2.4	Booleans.....	22
6.2.5	Listen.....	23
6.2.6	Schleifen.....	23
6.2.7	Dictionaries.....	23
6.2.8	Zeilenumbrüche.....	24
7	Variablen und Facts.....	25
7.1	Facts.....	25
7.1.1	Übung.....	25
7.2	Playbook für mehrere Distributionen.....	25
7.2.1	Variable für Web-Verzeichnis.....	27

7.3	Eingebaute Variablen.....	28
7.3.1	Übung.....	28
7.3.2	Lösung.....	29
7.4	Reihenfolge der Variablen.....	30
8	Inventory.....	31
8.1	Inventory-Datei.....	31
8.2	Gruppen.....	31
8.3	Aliases.....	32
8.4	Host- und Gruppen-Variablen.....	33
8.5	Dynamische Inventories.....	33
9	Rollen und Best-Practices.....	34
9.1	Rollen.....	35
9.1.1	Übung.....	35
9.1.2	Struktur von Rollen.....	35

## 2 Was ist Ansible

Ansible ist ein Tool zum Automatisieren von Standard-Aufgaben:

- Provisioning von Systemen
- Konfigurations-Management
- Applikations-Deployment
- Task-Automatisierung
- Dokumentation der Umgebung

Der große Vorteil von Ansible ist, dass am Zielsystem kein Agent benötigt wird, sondern vorhandene System-Tools verwendet werden, wie zum Beispiel:

- SSH (Linux, UNIX, Switches,...)
- WinRM (Windows)
- APIs (AWS, GCE, Azure,...)

Standardmäßig werden Tasks parallel auf mehreren Maschinen gleichzeitig abgearbeitet. Die Automatisierungssprache ist einfach zu erlernen, da sie im YAML-Format ist.

### 2.1 Ansible Engine vs. Ansible Tower

Ansible Engine bzw. Ansible Core ist eine einfache Automatisierungssprache, welche eine IT Applikations-Infrastruktur in Ansible Playbooks beschreiben kann.

Ansible Tower ist ein Enterprise Framework zur Verwaltung, Kontrolle und Absicherung der Ansible Automatisierung via Web-UI und Rest-API.

TOWER PROJECTS INVENTORIES TEMPLATES JOBS admin [Settings] [Menu] [Print] [Power]

DASHBOARD

30  
HOSTS

8  
FAILED HOSTS

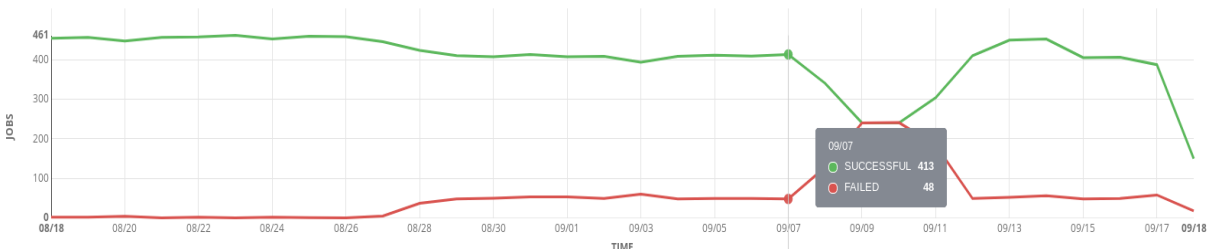
2  
INVENTORIES

0  
INVENTORY SYNC FAILURES

2  
PROJECTS

0  
PROJECT SYNC FAILURES

**JOB STATUS** PERIOD: PAST MONTH JOB TYPE: ALL VIEW: ALL



DATE	SUCCESSFUL	FAILED
09/07	413	48

**RECENTLY USED TEMPLATES** [VIEW ALL](#)

NAME	ACTIVITY	ACTIONS
Playbook: Bootstrap Development Systems	●●●●●●●●●●	
Playbook: Bootstrap Production Systems	●●●●●●●●●●	
Playbook: Update Linux hosts	●●	
Playbook: add ansible user	●●●●●●●●●●	
Demo: Create GCE webserver	●●●●●●●●●●	

**RECENT JOB RUNS** [VIEW ALL](#)

NAME	TIME
● Playbook: Bootstrap Development Systems	9/18/2017 10:34:59 AM
● Playbook: Bootstrap Production Systems	9/18/2017 10:33:38 AM
● Playbook: Bootstrap Development Systems	9/18/2017 10:04:53 AM
● Playbook: Bootstrap Production Systems	9/18/2017 10:03:33 AM
● Playbook: Bootstrap Development Systems	9/18/2017 9:34:43 AM

## 3 Testumgebung

Im ersten Teil des Workshops wird eine Testumgebung mit einem Ansible Host aufgebaut. Dieser Host ist sowohl Ansible Master als auch Ansible Client.

### 3.1 SSH

Damit für die Verbindung von Master auf Client kein Passwort eingegeben werden muss, wird ein SSH-Key für den root-Benutzer erstellt.

```
$ ssh-keygen -t rsa
```

Dieser Key wird nun dem root-Benutzer zugewiesen.

```
$ ssh-copy-id root@localhost
```

### 3.2 Übung

Um aktiv am Ansible Workshop teilnehmen zu können ist es nötig einen Ansible Master-Host sowie eine Test-VM zu besitzen. Die Test-VM ist in diesem Workshop gleichzeitig der Ansible Master.

Erstellen Sie eine Ansible Master VM (sofern Ihr Notebook/Workstation nicht mit Linux ausgestattet ist und Ansible darauf installiert werden kann) und konfigurieren Sie den lokalen SSH-Zugang.

## 4 Installation von Ansible

Nachdem ein Host, auf welchem Ansible laufen soll, bereit steht und eine Test-VM zum Testen der Playbooks vorhanden ist, wird im nächsten Schritt Ansible installiert.

### 4.1 Installation via Paket-Management-Tool

#### 4.1.1 RPM

Handelt es sich bei dem Host um ein Red Hat Enterprise Linux oder CentOS kann Ansible über das Extras- oder EPEL-Repository installiert werden. Bei CentOS ist das Extras-Repository bereits standardmäßig aktiv, bei Red Hat Enterprise Linux muss es via subscription-manager aktiviert werden. EPEL muss auf beiden Systemen aktiviert werden. Für diesen Workshop wird eine aktuelle Version von Ansible vorausgesetzt, welche sich in EPEL befindet.

```
$ sudo yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
$ sudo yum install ansible
```

Bei Fedora ist Ansible bereits in den offiziellen Repositories vorhanden.

```
$ sudo dnf install ansible
```

Für OpenSUSE wird zypper verwendet um Ansible aus den offiziellen Repositories zu installieren.

```
$ sudo zypper install ansible
```

#### 4.1.2 DEB

Sowohl Debian als auch Ubuntu bieten Pakete in ihren Repositories an.

```
$ sudo apt-get update
$ sudo apt-get install ansible
```

Da für diesen Workshop Ansible 2.5 (oder neuer) verwendet werden soll, müssen bei Debian und Ubuntu entsprechende Backport-Repositories eingerichtet werden oder Ansible mittels pip installiert werden.

#### 4.1.3 PIP

Bietet Ihre Distribution kein aktuelles Ansible an, so kann dies via pip installiert werden.

```
$ sudo apt-get install python-pip  
$ sudo pip install ansible
```

## 4.2 Übung

Installieren Sie nun auf dem Master-Host Ansible 2.5 (oder neuer) über einen der genannten Wege.



## 4.3 Ansible Test-VM einbinden

Damit Ansible Tasks in der Test-VM ausführen kann muss diese Ansible über ein Inventory(-File) bekannt gemacht werden. Nähere Details zu Inventories folgen später, für den ersten Schritt wird ein statisches Inventory-File namens hosts in einem Verzeichnis playbooks (welches in weiterer Folge alle Playbooks enthält) erzeugt.

```
$ mkdir playbooks
$ cd playbooks
$ vi hosts
```

Im Hosts-File wird sowohl ein Name (oder eine IP-Adresse sowie Verbindungsparameter angeben, sofern diese vom Standard abweichen).

IP-Adresse, Benutzer sowie Private Key werden nun in das hosts-File eingetragen.

```
$ vi hosts
testserver ansible_ssh_host=127.0.0.1 ansible_ssh_user=root
```

## 4.4 Verbindung testen

Das Überprüfen der Verbindung geht am einfachsten mittels des ansible ping-Kommandos.



Ansible versendet dabei keinen ping, sondern macht eine SSH-Verbindung.

```
$ ansible testserver -i hosts -m ping
testserver | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
```

## 4.5 Übung

Stellen Sie sicher, dass Ihre Test-VM vom Ansible-Host aus erreichbar ist.

## 5 Ad-Hoc Commands

Der einfachste Weg mittels Ansible Kommandos auf einer Vielzahl (oder einzelnen) Hosts auszuführen ist mittels Ad-Hoc Commands.

### 5.1 Einfache Kommandos

Einfache Liste aller aktuell verfügbaren Kommandos (Module) findet sich in der Ansible Dokumentation: [https://docs.ansible.com/ansible/list\\_of\\_all\\_modules.html](https://docs.ansible.com/ansible/list_of_all_modules.html)

Wichtig ist hier, dass bei den Modulen immer darauf geachtet wird, welche Ansible-Version mindestens nötig ist.

#### 5.1.1 Auslesen der Uptime.

```
$ ansible testserver -i hosts -m command -a uptime
testserver | SUCCESS | rc=0 >>
08:58:17 up 44 min, 1 user, load average: 0.00, 0.02, 0.0
```

Dabei wird das Module command aufgerufen und als Argument uptime übergeben.

#### 5.1.2 Log-Dateien betrachten

Das Modul command wird standardmäßig verwendet wenn man ansible aufruft, daher ist es ausreichend nur das Argument anzugeben. Erfolgt der Zugriff auf den Zielhost nicht als Benutzer root, muss für das Betrachten von Log-Dateien oder das Installieren von Paketen in der Regel sudo verwendet werden. Mit der Option "-b" wird die in ansible.cfg konfigurierte become-Methode (bei Linux sudo) verwendet.

```
$ ansible testserver -i hosts -a "tail /var/log/messages"
testserver | FAILED | rc=1 >>
tail: cannot open '/var/log/messages' for reading: Permission denied
```

#### 5.1.3 Paket installieren

Die Installation eines Pakets erfolgt mittels der Module yum/apt/zypper/... je nach Betriebssystem.

```
$ ansible testserver -m yum -a "name=httpd state=installed" -b
testserver | SUCCESS => {
  "changed": true,
  "msg": "",
  "rc": 0,
  "results": [
    "Loaded plugins: fastestmirror\nLoading mirror speeds from cached
hostfile\n * base: mirror.easynome.at\n * extras: mirror.easynome.at\n *
```

```

updates: mirror.easyname.at\nResolving Dependencies\n--> Running
transaction check\n--> Package httpd.x86_64 0:2.4.6-45.el7.centos will
be installed\n--> Processing Dependency: httpd-tools = 2.4.6-
45.el7.centos for package: httpd-2.4.6-45.el7.centos.x86_64\n-->
Processing Dependency: /etc/mime.types for package: httpd-2.4.6-
45.el7.centos.x86_64\n--> Processing Dependency: libaprutil-1.so.0()
(64bit) for package: httpd-2.4.6-45.el7.centos.x86_64\n--> Processing
Dependency: libapr-1.so.0()(64bit) for package: httpd-2.4.6-
45.el7.centos.x86_64\n--> Running transaction check\n--> Package
apr.x86_64 0:1.4.8-3.el7 will be installed\n--> Package apr-util.x86_64
0:1.5.2-6.el7 will be installed\n--> Package httpd-tools.x86_64 0:2.4.6-
45.el7.centos will be installed\n--> Package mailcap.noarch 0:2.1.41-
2.el7 will be installed\n--> Finished Dependency Resolution\n\
nDependencies Resolved\n\
n=====
=====\n Package                Arch          Version
Repository  Size\
n=====
=====\nInstalling:\n httpd                x86_64        2.4.6-
45.el7.centos      base          2.7 M\nInstalling for dependencies:\n
apr                x86_64        1.4.8-3.el7      base
103 k\n apr-util            x86_64        1.5.2-6.el7      base
92 k\n httpd-tools          x86_64        2.4.6-45.el7.centos      base
84 k\n mailcap              noarch        2.1.41-2.el7      base
31 k\n\nTransaction Summary\
n=====
=====\nInstall 1 Package (+4 Dependent packages)\n\nTotal download
size: 3.0 M\nInstalled size: 10 M\nDownloading packages:\
n-----
-----\nTotal                               819 kB/s |
3.0 MB 00:03 \nRunning transaction check\nRunning transaction test\
nTransaction test succeeded\nRunning transaction\n  Installing : apr-
1.4.8-3.el7.x86_64                               1/5 \n
Installing : apr-util-1.5.2-6.el7.x86_64
2/5 \n  Installing : httpd-tools-2.4.6-45.el7.centos.x86_64
3/5 \n  Installing : mailcap-2.1.41-2.el7.noarch
4/5 \n  Installing : httpd-2.4.6-45.el7.centos.x86_64
5/5 \n  Verifying   : httpd-tools-2.4.6-45.el7.centos.x86_64
1/5 \n  Verifying   : mailcap-2.1.41-2.el7.noarch
2/5 \n  Verifying   : apr-1.4.8-3.el7.x86_64
3/5 \n  Verifying   : httpd-2.4.6-45.el7.centos.x86_64
4/5 \n  Verifying   : apr-util-1.5.2-6.el7.x86_64
5/5 \n\n\nInstalled:\n httpd.x86_64 0:2.4.6-45.el7.centos
\n\nDependency Installed:\n apr.x86_64 0:1.4.8-3.el7
apr-util.x86_64 0:1.5.2-6.el7 \n httpd-tools.x86_64 0:2.4.6-
45.el7.centos mailcap.noarch 0:2.1.41-2.el7 \n\nComplete!\n"
]
}

```

## 5.2 Übung

Legen Sie einen Benutzer mit den folgenden Eigenschaften an:

- Name: testuser
- Gecos: "Test User"
- Shell: /bin/bash
- stellen Sie sicher, dass beim erstmaligen Login das Home-Directory (/home/testuser) vorhanden ist bzw. angelegt wird



[https://docs.ansible.com/ansible/list\\_of\\_all\\_modules.html](https://docs.ansible.com/ansible/list_of_all_modules.html)

oder:



```
$ ansible-doc
```

## 5.2.1 Lösung

Anzeigen der Informationen zum Modul user.

```
$ ansible-doc user
```

Anlegen des Benutzers.

```
$ ansible testserver -i hosts -m user -a "name=testuser comment=\"Test
User\" shell=/bin/bash state=present" -b
testserver | SUCCESS => {
  "append": false,
  "changed": true,
  "comment": "Test User",
  "group": 1001,
  "home": "/home/testuser",
  "move_home": false,
  "name": "testuser",
  "shell": "/bin/bash",
  "state": "present",
  "uid": 1001
}
```

## 6 Playbooks

Ad-hoc Kommandos sind nützlich wenn man schnell einen Befehl auf einer Vielzahl von Systemen ausführen muss (z.B. yum update), allerdings möchte man in der Regel diese auch wiederverwenden und die Bash-History hilft hier nur bedingt. Daher erstellt man in Ansible meistens Playbooks, welche eine Sammlung von Tasks enthalten.

In dieser Übung wird ein Webserver (Apache) installiert und eine Standard-Webseite ausgerollt. Dies soll die Einfachheit von Ansible verdeutlichen und wie rasch man zu einem Ergebnis kommen kann.

Um ein Playbook zu verstehen muss man wissen woraus dieses besteht.

# Playbook

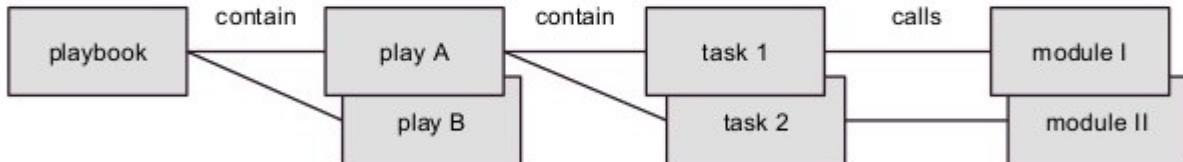


Illustration 1: Quelle: <https://image.slidesharecdn.com/ansible-150925121447-lva1-app6891/95/ansible-101-16-638.jpg?cb=1443183393>

Ein Playbook besteht aus einem oder mehreren **Plays**:

```
- name: Configure Apache webserver
```

Jedes Play besteht aus einem oder mehreren **Tasks**:

```
- name: Enable epel repository
- name: Install Apache
```

Jeder Task verwendet zur Ausführung ein **Modul**:

- yum
- copy
- template
- service

## 6.1 Einfaches Playbook

Im weiteren Fortschritt des Workshops wird auf Best-Practices zum Strukturieren der Playbooks, Rollen, Tasks,... eingegangen, im ersten Schritt verwenden wir ein einfaches File.

### 6.1.1 Anlegen eines Playbooks zum Installieren von Apache.



Ansible Playbooks sind im YAML-Format, daher muss die YAML-Syntax genau eingehalten werden und es dürfen z.B. nicht Spaces mit Tabulatoren gemeinsam verwendet werden!

```
$ vi web-notls.yml
---
- name: Configure Apache webserver
  hosts: webserver
  become: true
  tasks:
# für Red Hat basierte Systeme:
  - name: Install Apache
    yum:
      name: httpd
      state: present
# oder für Debian basierte Systeme:
  - name: Install Apache
    apt:
      name: apache2
      state: present
      update_cache: yes
# für SUSE basierte Systeme:
  - name: Install Apache
    zypper:
      name: apache2
      state: present
```



Beachten Sie die unterschiedliche Schreibweise/Syntax zwischen den Ad-hoc Kommando-Argumenten (=) vs. eines strukturierten Playbooks, bei welchem jedes Argument in einer eigenen Zeile steht (:).

Dieses Ansible-Playbook soll nur für Hosts der Gruppe webserver verwendet werden und als Benutzer root am Zielsystem ausgeführt werden. Um eine Gruppe webserver zu erhalten, wird diese im Inventory-File angelegt.

```
$ vi hosts

[webserver]
testserver ansible_ssh_host=127.0.0.1 ansible_ssh_user=root
```

Das Playbook wird nun nicht mittels des Kommandos ansible sondern mit ansible-playbook aufgerufen.

```
$ ansible-playbook -i hosts web-notls.yml
PLAY [Configure Apache webserver]
*****

TASK [setup]
*****
ok: [testserver]

TASK [Install Apache]
*****
changed: [testserver]

PLAY RECAP
*****
testserver          : ok=2    changed=1    unreachable=0
failed=0
```



## 6.1.2 Files deployen

Vielfach möchte man statische Dateien auf Systeme bekommen. In diesem Fall wird die statische HTML-Seite `static.html` auf den Host deployed.

```
$ vi web-notls.yml
...
# für Red Hat und Debian basierte Systeme
- name: Copy static.html
  copy:
    src: files/static.html
    dest: /var/www/html/static.html
# für SUSE basierte Systeme
- name: Copy static.html
  copy:
    src: files/static.html
    dest: /srv/www/htdocs/static.html
```

Statische Dateien sollten innerhalb eines Playbooks (bzw. einer Rolle, dazu aber später mehr) im Unterordner `files` liegen.

```
$ mkdir files
$ vi files/static.html
<html>
  <head>
    <title>Ansible workshop</title>
  </head>
  <body>
    <h1>Ansible workshop</h1>
    <p>This is our first web application, created in Ansible
workshop!</p>
  </body>
</html>
```

Deployen der statischen Konfigurations-Datei.

```
$ ansible-playbook -i hosts web-notls.yml
PLAY [Configure Apache webserver]
*****
TASK [setup]
```

```
*****
ok: [testserver]

TASK [Install Apache]
*****
ok: [testserver]

TASK [Copy static.html] *****
changed: [testserver]

PLAY RECAP
*****
testserver          : ok=3    changed=1    unreachable=0
failed=0
```



Beachten Sie, dass der bereits ausgeführte Task zum Installieren von Apache nicht erneut gemacht werden (**ok**), da diese Pakete bereits installiert sind.

### 6.1.3 Templates

Vielfach möchte man keine statischen Files, sondern dynamische Files, welche Variablen oder Schleifen enthalten können, auf das Zielsystem bekommen. Dazu kopieren wir nun ein Template mit einer Variable auf das Zielsystem.

```
$ vi web-notls.yml

...

# für Red Hat und Debian basierte Systeme

- name: Copy index.html
  template:
    src: templates/index.html.j2
    dest: /var/www/html/index.html
    mode: 0644

# für SUSE basierte Systeme

- name: Copy index.html
  template:
    src: templates/index.html.j2
    dest: /srv/www/htdocs/index.html
    mode: 0644
```



Templates werden nicht mittels der Moduls copy, sondern mit template definiert und liegen im Ordner templates!

```
$ mkdir templates  
$ vi templates/index.html.j2
```

Templates werden mittels der Templating-Engine Jinja2 erzeugt bzw. gerendert und tragen die Dateieindung .j2. Informationen dazu finden sich unter: <http://jinja.pocoo.org/docs/2.9/>



Es ist nicht nötig die Jinja-Dokumentation komplett zu studieren, da in der Regel nur sehr wenige Optionen verwendet werden (sollen und müssen).

```
<html>  
  <head>  
    <title>Ansible workshop</title>  
  </head>  
  <body>  
    <h1>Ansible workshop</h1>  
    <p>This is our first web application, created in Ansible  
workshop!</p>  
    <p>{{ ansible_managed }}</p>  
  </body>  
</html>
```

Damit die Webseite auch betrachtet werden kann, muss das Service Apache gestartet werden und die Firewall konfiguriert werden oder gestoppt sein.

```
$ vi web-notls.yml
...
# für Red Hat basierte Systeme
- name: Start and enable Apache
  service:
    name: httpd
    state: started
    enabled: yes

# für Debian und SUSE basierte Systeme
- name: Start and enable Apache
  service:
    name: apache2
    state: started
    enabled: yes

# für alle Systeme
- name: Stop and disable firewalld
  service:
    name: firewalld
    state: stopped
    enabled: no
    ignore_errors: yes
```

Mit der Option `ignore_errors` können Fehler ignoriert werden, z.B. wenn ein Dienst, welcher nicht laufen darf, gar nicht installiert ist.

Anschließend erfolgt wiederum ein Playbook Run.

```
$ ansible-playbook -i hosts web-notls.yml
...
TASK [Copy index.html]
*****
changed: [testserver]

TASK [Start and enable Apache]
*****
changed: [testserver]

TASK [Stop and disable firewalld]
*****
fatal: [testserver]: FAILED! => {"changed": false, "msg": "Could not find
the requested service firewalld: host"}
...ignoring
```

## 6.1.4 Handler

Ein Handler ist ähnlich wie ein Task, läuft aber nur dann, wenn er von einem Task via **notify** aufgerufen wird. Ein Anwendungsfall dafür ist zum Beispiel das Neustarten eines Diensts nach einer Änderung an einer Konfigurationsdatei.

Wir starten den Apache via Handler neu wenn sich das static.html File geändert hat. Dies ist in der Praxis natürlich nicht nötig, der Einfachheit halber wird diese Datei aber als Apache-Konfigurationsdatei betrachtet.

```
$ vi web-notls.yml
...
# für Red Hat und Debian basierte Systeme
- name: Copy index.html
  template:
    src: templates/index.html.j2
    dest: /var/www/html/index.html
    mode: 0644
  notify: Restart Apache

# für SUSE basierte Systeme
- name: Copy index.html
  template:
    src: templates/index.html.j2
    dest: /srv/www/htdocs/index.html
    mode: 0644
  notify: Restart Apache
...

# Für Red Hat basierte Systeme
handlers:
  - name: Restart Apache
    service:
      name: httpd
      state: restarted

# Für Debian und SUSE basierte Systeme
handlers:
  - name: Restart Apache
    service:
      name: apache2
      state: restarted
```

Der Handler wird nur aufgerufen wenn sich die Datei static.html am Ansible-Master geändert hat.

## 6.2 YAML-Syntax

Im Zuge des Workshops wurde bereits einiges an YAML-Code geschrieben. Als zukünftige Referenz wird nun auf Kommentare, Strings,... eingegangen.

### 6.2.1 Playbook Header

Ein Playbook sollte, damit es sich um korrekte YAML-Syntax handelt, immer mit

```
---
```

starten. Fehlen die 3 Minus-Zeichen ist dies Ansible egal, aber keine korrekte YAML-Syntax.

### 6.2.2 Kommentare

Kommentare werden wie im Perl oder Bash mit einer Raute (#) eingeleitet.

```
# This is a comment
```

### 6.2.3 String

Grundsätzlich müssen Strings nicht in Hochkomma (" bzw ') gesetzt werden, es ist aber auch nicht verboten. Eine Ausnahme bilden Variable in den Playbooks (nicht in den Templates wie bereits gesehen), diese müssen mit Hochkommata versehen werden:

```
- name: Install "{{ package_name }}"  
  yum: name="{{ package_name }}" state=installed
```

### 6.2.4 Booleans

Booleans sind aus YAML-Sicht folgende Werte und können genau so in einem Playbook verwendet werden:

- true / false
- True / False
- TRUE / FALSE
- yes / no
- Yes / No
- YES / NO
- on / off
- On / Off
- ON / OFF
- y / n
- Y / N

## 6.2.5 Listen

In YAML werden Listen durch - symbolisiert. Unser Playbook besteht also aus einer Liste von Plays (genau 1) sowie Tasks (mehrere).

```
- item 1
- item 2
- item 3
```

Möchte man mehrere Pakete installieren, kann man z.B. an das yum-Modul eine Liste übergeben:

```
yum:
  name:
    - httpd
    - mod_ssl
  state: present
```

## 6.2.6 Schleifen

Eine ausführliche Dokumenta-tion zu diesen Thema findet sich unter:

[https://docs.ansible.com/ansible/playbooks\\_conditionals.html](https://docs.ansible.com/ansible/playbooks_conditionals.html)

Die am häufigsten verwendete Schleife ist die with\_items (Standard)-Schleife. Möchte man mehrere Dateien auf ein Zielsystem mittels des copy-Moduls kopieren geht dies folgendermaßen:

```
copy:
  src: "{{ item }}"
  dest: "/var/www/html/{{ item }}"
with_items:
  - index.html
  - index2.html
```

item ist dabei eine spezielle Variable. Variablen werden im nächsten Kapitel genauer beschrieben.

Seit Ansible 2.5 gibt es die loop-Schleife, welche alle anderen with\_\* Schleifen ersetzen kann. Startet man neu mit Ansible kann man gleich mit loop starten. Eine Auflistung über alle möglichen Schleifen findet sich hier: [https://docs.ansible.com/ansible/latest/user\\_guide/playbooks\\_loops.html](https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html)

## 6.2.7 Dictionaries

Dictinaries entsprechen Hashes aus den meisten Programmiersprachen, welche durch eine Zuweisung eines Wertes an eine Variable mittels : definiert werden.

```
yum:
  name: httpd
  state: installed
```

## 6.2.8 Zeilenumbrüche

Zeilenumbrüche sind in YAML (bzw. Ansible) mittels `>` möglich.

```
yum: >
  name=httpd
  state=installed
```



## 7 Variablen und Facts

Bislang wurde nur die spezielle Variable `ansible_managed` in einem Template verwendet. In diesem Kapitel wird auf Variablen und Facts genauer eingegangen.

### 7.1 Facts

Eine spezielle Form der Variablen sind die Facts. Vermutlich ist beim Starten der Playbooks bereits aufgefallen, dass ein Task ausgeführt wird, welcher in keinem Playbook definiert ist:

```
$ ansible-playbook -i hosts web-tls.yml

PLAY [Configure Apache webserver]
*****

TASK [setup]
*****
ok: [testserver]
```

Der Task `setup` sammelt Fakten über das System, welche er anschließend verwendet:

```
$ ansible -i host -m setup testserver
testserver | SUCCESS => {
  "ansible_facts": {
    "ansible_all_ipv4_addresses": [
      "192.168.121.65"
    ],
    "ansible_all_ipv6_addresses": [
      "fe80::5054:ff:fede:7d69"
    ],
    "ansible_architecture": "x86_64",
    "ansible_bios_date": "04/01/2014",
    "ansible_bios_version": "1.9.3-1.fc25",
    ...
  }
}
```

#### 7.1.1 Übung

Analysieren Sie welche Informationen über Ihren Testserver gesammelt werden.

## 7.2 Playbook für mehrere Distributionen

Durch das gesammelte Wissen über Facts können wir das Playbook für mehrere Distributionen verfügbar machen.



Es wären noch mehr Änderungen am Playbook nötig (Services, Handlers), in der Kürze des Workshops wird allerdings nur beispielhaft die Installation gezeigt.

```
$ vi web-tls.yml
...
tasks:
  - name: Install Apache on RHEL
    yum:
      name: httpd
      state: present
      when: ansible_os_family == "RedHat"
  - name: Install Apache on Debian
    apt:
      name: apache2
      state: present
      update_cache: yes
      when: ansible_os_family == "Debian"
  - name: Install Apache on SUSE
    zypper:
      name: apache2
      state: present
      when: ansible_os_family == "Suse"
  - name: Copy static.html
...

$ ansible-playbook -i hosts web-notls.yml
...
TASK [Install Apache on RHEL]
*****
skipping: [testserver]

TASK [Install Apache on Debian]
*****
skipping: [testserver]

TASK [Install Apache on SUSE]
*****
```

```
*****  
ok: [testserver]  
...  
...
```

## 7.2.1 Variable für Web-Verzeichnis

Das Apache Web-Verzeichnis unterscheidet sich bei Red Hat sowie Debian grundlegend von SUSE. Hier könnte man das Verzeichnis als Variable definieren und pro System überschreiben.

```
$ vi web-notls.yml  
  
---  
- name: Configure Apache webserver  
  hosts: webservers  
  become: true  
  
# für Red Hat und Debian Systeme  
vars:  
  apache_webroot: /var/www/html  
  
# für SUSE Systeme  
vars:  
  apache_webroot: /srv/www/htdocs  
  
tasks:  
  
...  
  
- name: Copy static.html  
  copy:  
    src: files/static.html  
    dest: "{{ apache_webroot }}/static.html"  
  notify: Restart Apache  
  
- name: Copy index.html  
  template:  
    src: templates/index.html.j2  
    dest: "{{ apache_webroot }}/index.html"  
    mode: 0644  
  notify: Restart Apache  
  
...  
  
$ ansible-playbook web-notls.yml  
...  
  
TASK [Copy static.html]  
*****  
*****
```

```
ok: [testserver]

TASK [Copy index.html]
*****
*****
ok: [testserver]

...

```

## 7.3 Eingebaute Variablen

Ansible bringt auch einige eingebaute Variablen mit wie:

- `hostvars`: Liste mit den Hostvariablen eines (oder mehrerer) Hosts
- `inventory_hostname`: der Name des Hosts im Ansible Inventory
- `group_names`: Liste der Gruppen in welcher ein Host ist
- `groups`: Liste aller Gruppen und Mitglieder
- `play_hosts`: Liste der Hosts im aktuellen Play
- `ansible_version`: Ansible Version

### 7.3.1 Übung

Passen Sie das `web-notls` Playbook an, dass Ihre Webseite den folgenden Inhalt anzeigt:

```
This is our first web application on $HOSTNAME, created in Ansible workshop!
```

Ersetzen Sie `$HOSTNAME` dabei durch den Namen des Hosts im Ansible Inventory.

## 7.3.2 Lösung

Hierfür muss nur das Template `index.html.j2` um die Variable `inventory_hostname` erweitert werden.

```
$ vi templates/index.html.j2
...
                <h1>Ansible workshop</h1>
                <p>This is our first web application on {{ inventory_hostname
}}, created in Ansible workshop!</p>
                <p>{{ ansible_managed }}</p>
...

$ ansible-playbook -i hosts web-notls.yml
...

TASK [Copy index.html]
*****
changed: [testserver]

...
```

## 7.4 Reihenfolge der Variablen

Seit Ansible 2 gilt die folgende Reihenfolge, wobei das 1. Listing das schwächste ist und das letzte das Stärkste (siehe [https://docs.ansible.com/ansible/playbooks\\_variables.html](https://docs.ansible.com/ansible/playbooks_variables.html)):

1. role defaults
2. inventory vars
3. inventory group\_vars
4. inventory host\_vars
5. playbook group\_vars
6. playbook host\_vars
7. host facts
8. play vars
9. play vars\_prompt
10. play vars\_files
11. registered vars
12. set\_facts
13. role and include vars
14. block vars (only for tasks in block)
15. task vars (only for the task)
16. extra vars (always win precedence)

Ein sinnvolles Setzen von Variablen wäre zum Beispiel:

1. Defaults in einer Rolle
2. (Inventory) Gruppen-Variablen
3. (Inventory) Host-Variablen

So ist man in der Lage Standardwerte in einer Rolle (was das ist später) zu setzen aber die Möglichkeit offen zu lassen diese über Inventory Gruppen- sowie Host-Variablen zu überschreiben.

## 8 Inventory

Das Inventory enthält eine Auflistung aller Hosts, die Ansible bedienen kann bzw. soll sowie eine Unterteilung in Gruppen.

### 8.1 Inventory-Datei

Die Inventory-Datei, welche im Zuge des Workshops erstellt wurde, enthält bislang 1 Host und 1 Gruppe.

```
$ vi hosts  
  
[webservers]  
testserver ansible_ssh_host=127.0.0.1 ansible_ssh_user=root
```

### 8.2 Gruppen

Im Ansible Inventory können (und sollen) mehrere Gruppen definiert sein. Ein Host kann Mitglied in mehreren Gruppen sein. Dies ist vor allem dann sinnvoll, wenn Variablen auf Gruppen-Ebene gesetzt werden.

Die spezielle Gruppe **all** wird im Inventory nicht definiert, sondern enthält alle Hosts des Inventories. Gruppen werden immer mittels des [-Zeichens definiert und können auch verschachtelt (:**children**) sein.

```
$ vi hosts  
  
[development:children]  
devwebservers  
  
[devwebservers]  
testserver ansible_ssh_host=127.0.0.1  
  
[production]  
prod01.example.com  
prod02.example.com  
  
[production:children]  
prodwebservers  
  
[prodwebservers]  
web01.example.com  
web02.example.com  
web03.example.com
```

Wie im Beispiel oben ersichtlich ist eine Logik in der Benennung zu erkennen (aufsteigende Nummerierung), welche sich im Inventory auch vereinfachen lässt:

```
$ vi hosts

[devwebservers]
testserver ansible_ssh_host=127.0.0.1

[production]
prod0[1-2].example.com

[prodwebservers]
web0[1-3].example.com
Dies geht auch mit Buchstaben.
z.B. web-[a-f].example.com
```



Dies geht auch mit Buchstaben.  
z.B. web-[a-f].example.com

## 8.3 Aliases

Ansible versucht immer den Hostnamen des Hosts im Inventory aufzulösen und sich darauf zu verbinden. Ist dies wie im obigen Beispiel nicht möglich, da der testserver nicht auflösbar ist, so wird ein Alias vergeben und die Verbindungsdetails mittels `ansible_ssh_host` mitgeteilt.

```
$ vi hosts

[webservers]
testserver ansible_ssh_host=127.0.0.1 ansible_ssh_user=root
```



Seit Ansible 2.0 sollte man `ansible_host` anstelle von `ansible_ssh_host` (genau wie für `port`, `user`,...) verwenden. Versionen < 2.0 kennen `ansible_host` nicht, daher verwenden wir dies noch wegen der Rückwärtskompatibilität.



## 8.4 Host- und Gruppen-Variablen

Wie bereits erwähnt verwendet man Gruppen vielfach um Variablen zu setzen sowie Playbooks nur für eine bestimmte Gruppe von Hosts laufen zu lassen. Variablen können direkt im Inventory-File definiert werden oder besser in eigenen Dateien.

```
$ vi hosts

[all:vars]
nginx_services:
- http
- https
nginx_ports:
- 80
- 443
nginx_key: localhost.key
nginx_cert: localhost.crt

[development:children]
webservers

[webservers]
testserver ansible_ssh_host=192.168.121.65
```

Besser und übersichtlicher ist es natürlich wenn die Variablen in einem eigenen Ordner liegen. Hierfür sind die folgenden Ordner vorgesehen:

- group\_vars - für Gruppen-Variablen
- host\_vars - für Host-Variablen

## 8.5 Dynamische Inventories

Bislang ist das Inventory, welches erstellt wurde rein statisch. Dies setzt voraus, dass man alle Hosts kennt, welche mit Ansible verwaltet werden sollen und dass jemand diese Datei pflegt. Gerade in größeren Umgebungen ist entweder eine Asset-Datenbank oder CMDB vorhanden, welche alle Hosts enthält oder man möchte alle Instanzen aus einer Cloud-Umgebung auslesen. Dafür gibt es die Möglichkeit das Inventory dynamisch durch Inventory-Skripte bei der Laufzeit von Ansible zu erstellen.

Bereits vorhandene Inventory-Skripte finden sich hier:

<https://github.com/ansible/ansible/tree/devel/contrib/inventory>

Alle Inventory-Skripte müssen die folgenden Optionen kennen (falls Sie ein eigenes Skript schreiben möchten):

- --host=<hostname>: Um Informationen über einen bestimmten Host auszulesen
- --list: Zum Auflisten aller Gruppen

## 9 Rollen und Best-Practices

In diesem Teil des Workshops geht es darum zu verdeutlichen wie man Playbooks in Rollen aufteilt und welche Best-Practices es gibt um Playbooks zu strukturieren. Die wichtigste Seite in diesem Zusammenhang ist:

[https://docs.ansible.com/ansible/playbooks\\_best\\_practices.html](https://docs.ansible.com/ansible/playbooks_best_practices.html)

Hier wird auch verdeutlicht wie ein Playbook am besten strukturiert sein sollte:

```
production          # inventory file for production servers
staging             # inventory file for staging environment

group_vars/
  group1            # here we assign variables to particular groups
  group2            # ""
host_vars/
  hostname1         # if systems need specific variables, put them here
  hostname2         # ""

library/            # if any custom modules, put them here (optional)
filter_plugins/    # if any custom filter plugins, put them here
(optional)

site.yml            # master playbook
webservers.yml     # playbook for webserver tier
dbservers.yml      # playbook for dbserver tier

roles/
  common/           # this hierarchy represents a "role"
    tasks/         #
      main.yml      # <-- tasks file can include smaller files if
warranted
  handlers/        #
    main.yml       # <-- handlers file
  templates/       # <-- files for use with the template resource
    ntp.conf.j2    # <----- templates end in .j2
  files/           #
    bar.txt        # <-- files for use with the copy resource
    foo.sh         # <-- script files for use with the script
resource
  vars/            #
    main.yml       # <-- variables associated with this role
  defaults/       #
    main.yml       # <-- default lower priority variables for this
role
  meta/           #
    main.yml      # <-- role dependencies
  library/        # roles can also include custom modules
  lookup_plugins/ # or other types of plugins, like lookup in this
case
```

```
webtier/           # same kind of structure as "common" was above,  
done for the webtier role  
monitoring/       # ""  
fooapp/           # ""
```

## 9.1 Rollen

Rollen dienen dazu ein Playbook in mehrere Dateien aufzuteilen. Weiters können Rollen von beliebigen Playbooks wiederverwendet werden.

Z.B. Sie erstellen eine Rolle `mysql` zur Konfiguration einer MySQL-Datenbank und verwenden diese in den Playbooks für Wordpress und Intranet.

Es gibt bereits fertige Rollen von Community-Mitglieder, welche auf Ansible Galaxy gesammelt werden.

### 9.1.1 Übung

Verschaffen Sie sich einen Überblick über bereits existierende Rollen:

[https://galaxy.ansible.com/list#/roles?page=1&page\\_size=10](https://galaxy.ansible.com/list#/roles?page=1&page_size=10)

Selbst wenn man keine fertigen Rollen verwenden möchte sind diese am Anfang teilweise sehr hilfreich um im Ansible-Code nachsehen zu können wie andere ein Problem gelöst haben.

### 9.1.2 Struktur von Rollen

Rollen bestehen aus Tasks, Files, Templates, Variablen, Handlern, Meta-Daten und Defaults. Viele dieser Ordner sind bereits aus dem Playbook bekannt. Neu ist hingegen, dass die Tasks in einem eigenen Ordner wandern und im Playbook keine Tasks sondern Rollen aufgerufen werden.

Neue Rollen lassen sich mit dem Command **ansible-galaxy** erstellen (unabhängig davon ob die Rollen jemals auf Ansible Galaxy veröffentlicht werden oder nicht). Das Tool erstellt dabei die korrekten Ordner-Strukturen:

```
$ ansible-galaxy init -p roles webserver  
- webserver was created successfully  
$ find roles/webserver/  
roles/webserver/  
roles/webserver/.travis.yml  
roles/webserver/meta  
roles/webserver/meta/main.yml  
roles/webserver/handlers  
roles/webserver/handlers/main.yml  
roles/webserver/tasks  
roles/webserver/tasks/main.yml  
roles/webserver/templates  
roles/webserver/vars  
roles/webserver/vars/main.yml  
roles/webserver/files
```

```
roles/webserver/tests
roles/webserver/tests/inventory
roles/webserver/tests/test.yml
roles/webserver/defaults
roles/webserver/defaults/main.yml
roles/webserver/README.md
```