

# **Terraform & Ansible Grundlagen**

B1 Systems GmbH

05.-09. Februar 2024



If you have any questions, comments or want to report errors in the training material please post them to [doku@b1-systems.de](mailto:doku@b1-systems.de).

Please note that all soft and hardware names, trademarks and product names of the respective firms used in this manual remain property of their holders even if not marked accordingly.

© B1 Systems GmbH 2004 – 2024; Course materials may not be reproduced in whole or in part without the written permission of B1 Systems.

# Inhaltsverzeichnis

<b>0 Typographische Konventionen</b>	<b>1</b>
<b>1 Grundlagen des Cloud Computing</b>	<b>5</b>
1.1 Zielsetzung . . . . .	6
1.2 Cloud Computing . . . . .	7
1.3 Was ist OpenStack? . . . . .	14
1.3.1 Komponenten von OpenStack . . . . .	16
<b>2 Einführung in Terraform</b>	<b>19</b>
2.1 Zielsetzung . . . . .	20
2.2 Was ist Terraform? . . . . .	21
2.3 Infrastructure as Code (IaC) . . . . .	25
2.4 Provider . . . . .	27
2.5 Einsatzbereiche . . . . .	29
2.6 Alternativlos? . . . . .	30
2.7 Terraform und Vagrant . . . . .	32
2.8 Terraform Enterprise . . . . .	33
2.9 Links und Quellen . . . . .	34
<b>3 Installation von Terraform</b>	<b>35</b>
3.1 Zielsetzung . . . . .	36
3.2 Terraform installieren . . . . .	37
3.3 Aufgabenteil . . . . .	39
<b>4 Terraform CLI – Grundlegende Befehle</b>	<b>41</b>
4.1 Zielsetzung . . . . .	42
4.2 Terraform CLI . . . . .	43
4.3 Workspace anlegen . . . . .	45
4.4 Terraform Settings Block . . . . .	47
4.5 Aufgabenteil . . . . .	49
<b>5 Terraform – Provider</b>	<b>51</b>
5.1 Zielsetzung . . . . .	52
5.2 Funktionen eines Providers . . . . .	53
5.3 Beispiel – OpenStack . . . . .	54
5.4 Beispiel – AWS . . . . .	55
5.5 Beispiel – GCP . . . . .	56
5.6 Beispiel – Azure . . . . .	57
5.7 Weitere Provider . . . . .	58
5.8 Arbeiten ohne Provider im Code . . . . .	59
5.9 Aufgabenteil . . . . .	61
<b>6 Terraform – Ressourcen erzeugen</b>	<b>63</b>
6.1 Zielsetzung . . . . .	64
6.2 Compute Ressource erzeugen . . . . .	65
6.3 Aufgabenteil . . . . .	67

## Inhaltsverzeichnis

6.4 Mehrere Compute Ressourcen erzeugen . . . . .	72
6.5 Aufgabenteil . . . . .	73
6.6 Modifizieren von Ressourcen . . . . .	80
6.7 Aufgabenteil . . . . .	81
6.8 Netzwerk Ressource hinzufügen . . . . .	99
6.9 Aufgabenteil . . . . .	101
6.10 Floating IP Ressource . . . . .	116
6.11 Aufgabenteil . . . . .	119
<b>7 Terraform – Umgang mit Variablen und Outputs</b>	<b>125</b>
7.1 Zielsetzung . . . . .	126
7.2 Data Types . . . . .	127
7.3 Variablen deklarieren und initialisieren . . . . .	129
7.4 Outputs . . . . .	133
7.5 Umstellung auf Variablen und Outputs . . . . .	136
7.6 Aufgabenteil . . . . .	139
7.7 Abfragen von Daten . . . . .	146
7.8 Aufgabenteil . . . . .	147
7.9 Block Devices . . . . .	149
7.10 Aufgabenteil . . . . .	151
7.11 null Ressourcen . . . . .	164
7.12 Aufgabenteil . . . . .	167
<b>8 Terraform – Speichern von Zuständen</b>	<b>170</b>
8.1 Zielsetzung . . . . .	171
8.2 <code>terraform.tfstate</code> . . . . .	172
<b>9 Terraform – Module</b>	<b>174</b>
9.1 Zielsetzung . . . . .	175
9.2 Aufbau von Modulen . . . . .	176
9.3 Module – Best Practices . . . . .	177
9.4 Aufgabenteil . . . . .	179
<b>10 Terraform – Arbeiten im Team</b>	<b>181</b>
10.1 Zielsetzung . . . . .	182
10.2 Git, SVN & Co . . . . .	183
10.3 Wrapper Scripts . . . . .	184
10.4 Admin Workstation . . . . .	185
<b>11 Terraform – Alternative Technologien</b>	<b>186</b>
11.1 Zielsetzung . . . . .	187
11.2 Ansible, Chef, Puppet, usw. . . . .	188
11.3 Libraries . . . . .	189
11.4 Cloudformation, Deployment Manager, Heat . . . . .	190
11.5 Pulumi . . . . .	191
<b>12 Einführung in Ansible</b>	<b>192</b>
12.1 Was ist Ansible? . . . . .	193
12.2 Einsatzgebiete . . . . .	195

12.3 Warum automatisiertes Deployment/Konfigurationsmanagement? . . . . .	202
12.4 Weitere Tools im Baukasten . . . . .	207
12.5 Alternativen zu Ansible . . . . .	208
<b>13 Installation von Ansible</b>	<b>216</b>
13.1 Zielsetzung . . . . .	217
13.2 Installation über Distributionspakete . . . . .	218
13.3 Aufgabenteil . . . . .	225
<b>14 Ansible Ad-Hoc Befehle</b>	<b>228</b>
14.1 Zielsetzung . . . . .	229
14.2 Ad-Hoc-Befehle . . . . .	230
14.2.1 Aufgabenteil . . . . .	237
14.3 Inventory . . . . .	239
14.3.1 Aufgabenteil . . . . .	241
14.3.2 SSH-Keys . . . . .	243
14.3.3 Aufgabenteil . . . . .	245
14.3.4 Gruppen im Inventory . . . . .	247
14.3.5 Aufgabenteil . . . . .	249
14.4 Targeting-Optionen für Ad-Hoc Befehle . . . . .	251
14.5 Ansible-Module . . . . .	254
14.5.1 Aufgabenteil . . . . .	259
14.6 Ansible Collections . . . . .	264
14.6.1 Aufgabenteil . . . . .	275
<b>15 Ansible Playbooks</b>	<b>278</b>
15.1 Zielsetzung . . . . .	279
15.2 Playbooks . . . . .	280
15.2.1 Aufgabenteil . . . . .	287
15.3 Mehrere Tasks in einem Playbook . . . . .	289
15.3.1 Aufgabenteil . . . . .	291
15.4 Variablen . . . . .	294
15.4.1 Variablen im Inventory . . . . .	295
15.4.2 Aufgabenteil . . . . .	299
15.4.3 host_vars und group_vars . . . . .	301
15.4.4 Aufgabenteil . . . . .	305
15.4.5 Ansible Facts . . . . .	307
15.4.6 Variablen im Playbook . . . . .	312
15.4.7 Dictionaries & Listen in YAML . . . . .	314
15.4.8 Registrierte Variablen . . . . .	321
15.4.9 Aufgabenteil . . . . .	323
15.4.10 Variablen auf der Kommandozeile . . . . .	326
15.5 Aufgabenteil . . . . .	331
15.6 When . . . . .	335
15.6.1 Aufgabenteil . . . . .	349
15.7 loop . . . . .	358
15.7.1 Aufgabenteil . . . . .	369

## Inhaltsverzeichnis

<b>16 Ansible Playbooks und Roles</b>	<b>373</b>
16.1 Zielsetzung . . . . .	374
16.2 Roles . . . . .	375
16.3 Struktur von Rollen . . . . .	377
16.3.1 Aufgabenteil . . . . .	379
16.3.2 Rollen: tasks . . . . .	383
16.3.3 Aufgabenteil . . . . .	387
16.3.4 Rollen: handlers . . . . .	389
16.3.5 Aufgabenteil . . . . .	399
16.3.6 Rollen: vars, defaults . . . . .	401
16.3.7 Aufgabenteil . . . . .	405
16.3.8 Rollen: vars . . . . .	407
16.3.9 Aufgabenteil . . . . .	413
16.3.10 Jinja2-Syntax . . . . .	416
16.3.11 Aufgabenteil . . . . .	423
16.3.12 Jinja2-Syntax: Kontrollstrukturen . . . . .	426
16.3.13 Aufgabenteil . . . . .	435
16.3.14 Jinja2-Syntax: Filter & Tests . . . . .	441
16.3.15 Aufgabenteil . . . . .	449
16.3.16 Rollen: templates/ . . . . .	454
16.3.17 Aufgabenteil . . . . .	459
16.3.18 Rollen: templates/ und tasks/ . . . . .	463
16.3.19 Aufgabenteil . . . . .	465
16.4 Aufruf von Rollen . . . . .	467
16.4.1 Aufgabenteil . . . . .	469
16.5 Aufruf mit Übergabe von Parametern . . . . .	472
16.5.1 Aufgabenteil . . . . .	475
16.6 Ansible Galaxy – Rollen . . . . .	477
16.6.1 Aufgabenteil . . . . .	483
16.7 Ansible Galaxy – Collections . . . . .	486
16.7.1 Aufgabenteil . . . . .	493
<b>17 Ansible Advanced</b>	<b>495</b>
17.1 Zielsetzung . . . . .	496
17.2 Ansible – Konfiguration . . . . .	497
17.2.1 Aufgabenteil . . . . .	499
17.3 Ansible Vault . . . . .	501
17.3.1 Aufgabenteil . . . . .	511
17.4 Delegation . . . . .	515
17.5 Advanced Inventory . . . . .	517
17.5.1 Andere Formate im Inventory . . . . .	518
17.5.2 Dynamische Inventories . . . . .	525
17.6 Lookups . . . . .	529
17.7 Set Fact . . . . .	532
17.8 Magische Variablen . . . . .	535
17.8.1 Aufgabenteil . . . . .	539
17.9 Advanced Loops . . . . .	541
17.9.1 Aufgabenteil . . . . .	545

17.10 include_* und import_* . . . . .	548
17.11 Blocks . . . . .	550
17.12 Privilege Escalation . . . . .	559
17.13 Ansible Tagging . . . . .	565
17.13.1 Aufgabenteil . . . . .	577
17.14 Strategies . . . . .	582
17.14.1 Strategy . . . . .	583
17.14.2 Serial . . . . .	590
17.14.3 Aufgabenteil . . . . .	597
17.15 Error Handling – Übersicht . . . . .	602
<b>18 Ansible im Team</b> . . . . .	<b>607</b>
18.1 Zielsetzung . . . . .	608
18.2 Ansible im Team . . . . .	609
18.3 Ansible im Team mit Git . . . . .	610
18.3.1 Einrichten von Gitea . . . . .	613
18.4 Ansible mit AWX . . . . .	623
18.5 Aufgabenteil . . . . .	645
18.6 Ansible Tower . . . . .	654



## 0 Typographische Konventionen



# Typographische Konventionen

# Typographische Konventionen

## Typographische Konventionen

Element	Auszeichnung
Datei- und Verzeichnisname	/etc/passwd
Benutzername	tux
URL	http://www.b1-systems.de
Variable	PATH
Platzhalter/Variable	<i>dateiname</i> oder <dateiname>
Kommando	ls -l
Menüpunkt/Eingabefeld/Button	OK
Taste bzw. Tastenkombination	<i>Alt+F1</i>
Hervorhebung	<i>Wichtige Information</i>

Die folgende Liste führt die wichtigsten typographischen Elemente auf, die in dieser Schulungsunterlage verwendet werden:

**/etc/passwd** Datei- und Verzeichnisnamen werden als /pfad/zur/datei gekennzeichnet.

**tux** Benutzernamen werden genau so wie Datei- und Verzeichnisnamen hervorgehoben:  
tux.

**http://www.b1-systems.de** URL.

**PATH** Angabe einer Variable mit PATH.

**dateiname | <dateiname>** Ein Platzhalter ist entweder durch Kursivschrift (PATH=*dateiname*) oder durch den Einschluss in spitze Klammern (PATH=<*dateiname*>) gekennzeichnet.

**ls -l** Kommandos werden wie Datei- und Verzeichnisnamen und Benutzernamen hervorgehoben.

**OK** Menüpunkt/Eingabefeld/Button.

**Alt+F1** Tastatureingabe bzw. Tastenkombination.

**Wichtige Information** Hervorhebung wichtiger Informationen.



# Shell-Prompt

Unterschiedliche Prompts für Superuser `root` und „gewöhnliche“ Benutzer:

Aufruf eines Befehls als „gewöhnlicher“ Benutzer

```
$ rpm -qi bash
```

Aufruf eines Befehls als `root`

```
# rpm -Uvh bash-<version>.rpm
```

Zur Kennzeichnung, ob ein Kommando von einem „gewöhnlichen“ Benutzer ausgeführt werden kann, oder ob nur der Superuser `root` dies darf, werden unterschiedliche Prompts in der Shell verwendet:

- Aufruf eines Befehls als „gewöhnlicher“ Benutzer:

```
$ rpm -qi bash
```

Das Zeichen `$` ist der Prompt für einen „gewöhnlichen“ Benutzer.

- Aufruf eines Befehls als `root`:

```
# rpm -Uvh bash-<version>.rpm
```

Das Zeichen `#` steht für den Prompt von `root`.



## 1 Grundlagen des Cloud Computing



# Grundlagen des Cloud Computing

## 1.1 Zielsetzung



# Zielsetzung

Dieses Kapitel

- liefert einen Überblick über „Cloud Computing“
- stellt OpenStack und seine einzelnen Komponenten kurz vor

## 1.2 Cloud Computing



# Cloud Computing

- Cloud Computing beschreibt ein Konzept, IT-Infrastruktur/Ressourcen abstrahiert über ein Netzwerk anzubieten
- Mögliche Ressourcen sind z. B. Rechenleistung, Speicher, Netzwerke, Serverdienste, Anwendungen
- Ressourcen können zur Laufzeit dynamisch erweitert oder reduziert werden
- offizielle Beschreibung des NIST (*National Institute of Standards and Technology*):
  - drei Servicemodelle (*Service Models*)
  - vier Liefermodelle (*Deployment Models*)
  - fünf Eigenschaften (*Essential Characteristics*)

Cloud Computing beschreibt ein Konzept, bei dem IT-Infrastruktur bzw. Ressourcen abstrahiert über ein Netzwerk angeboten werden.

Die im Rahmen von Cloud Computing angebotenen Ressourcen umfassen nahezu das komplette Spektrum der Informationstechnik – so etwa eine Infrastruktur (z. B. Rechenleistung, Datenspeicher, Netzwerke), aber auch Serverdienste, Anwendungen oder vorkonfigurierte Entwicklungsplattformen. Der Zugriff auf diese Ressourcen erfolgt über definierte technische Schnittstellen und Protokolle. Die Nutzung der Ressourcen kann dynamisch an den jeweiligen Bedarf angepasst werden, im einfachsten Fall mit minimalem Aufwand und ohne Interaktion mit einem Service Provider.

Aus Nutzersicht wird die Umgebung nicht mehr selbst betrieben oder örtlich bereitgestellt, sondern die zur Verfügung gestellte IT-Infrastruktur erscheint weit weg und un durchsichtig, so als wären die Ressourcen wie in einer „Wolke“ (engl. cloud) (ein abstrakter Wolkenumriss bildet in Netzwerkdiagrammen auch einen nicht genau umrissen Teil des Internets ab).

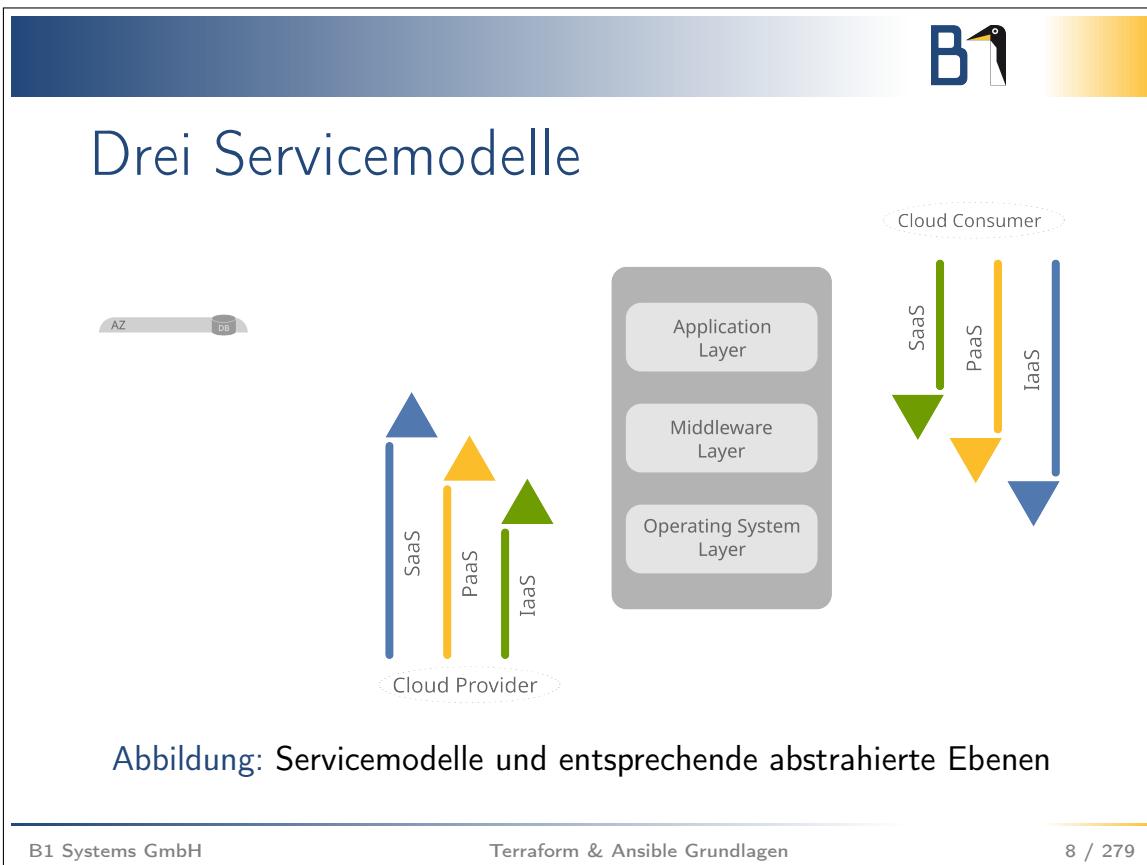
Der Zugriff auf die entfernten Systeme erfolgt über ein Netzwerk, beispielsweise über das Internet, oder auch über ein firmeninternes Intranet.

## 1 Grundlagen des Cloud Computing

Dabei kann der so genannte „Pooling-Effekt“ genutzt werden, der aus der gemeinsamen Nutzung von Ressourcen entsteht: Statt benötigte Ressourcen bei Bedarf jedes Mal neu bereitzustellen und danach wieder zu verwerfen, wird ein „Pool“ vorinitialisierter Ressourcen, die jederzeit einsatzbereit sind, vorgehalten.

Die offizielle Beschreibung von Cloud Computing des NIST (*National Institute of Standards and Technology*) beinhaltet drei Servicemodelle (*Service Models*), vier Liefermodelle (*Deployment Models*) und fünf Eigenschaften (*Essential Characteristics*). Detaillierte Informationen zu dieser Definition finden sich unter

<http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>.



Gemäß der offiziellen Beschreibung von Cloud Computing des NIST gibt es drei Servicemodelle (*Service Models*), die beschreiben, auf welcher Ebene eine Cloud ihre Dienste anbietet:

### Software as a Service (SaaS)

SaaS stellt dem Benutzer Software-Sammlungen und einzelne Applikationen zur Verfügung, die auf der Infrastruktur des Diensteanbieters laufen (*Application Layer*). Dabei kann es sich zum Beispiel um eine Webmailer-Oberfläche handeln. SaaS wird auch als *Software on demand* bezeichnet.

### Platform as a Service (PaaS)

PaaS stellt Nutzern eine Plattform bereit, die Programmierungs- oder Laufzeitumgebungen mit flexiblen, dynamisch anpassbaren Rechen- und Datenkapazitäten bieten. Es kann sich dabei zum Beispiel um eine vollständige IDE zur Java-Entwicklung handeln. Nutzer entwickeln ihre eigenen Software-Anwendungen oder lassen diese innerhalb einer Software-Umgebung, die vom Diensteanbieter (Service Provider) bereitgestellt und unterhalten wird, ausführen. PaaS ist der Ansatzpunkt für *Middleware*. Middleware stellt als Standard-Software Schnittstellen oder Dienste bereit, mit deren Hilfe Software-Komponenten plattformunabhängig kommunizieren können, wie dies beispielsweise bei verteilten Anwendungen gebraucht wird. Beispiele für Middleware verschiedener Hersteller:

## 1 Grundlagen des Cloud Computing

- *Common Object Request Broker Architecture (CORBA)* der Object Management Group (OMG)
- *D-Bus* vom freedesktop.org Projekt
- *Enterprise Service Bus* und *Fusion* von Oracle
- *JBoss Enterprise Application Platform* von Red Hat
- *SAP Exchange Infrastructure*
- *Transparent Inter Process Communication* (Tibco)
- *WebSphere Application Server (JavaEE Server)* von IBM

Mit dem darunter liegenden Betriebssystem und der Netzwerkkonfiguration kommen die Benutzer nicht in Berührung.

### **Infrastructure as a Service (IaaS)**

IaaS meint die Bereitstellung einer vollständigen Infrastruktur in virtualisierter Computer-Hardware mit Rechnern, Netzwerken und Speicher als Ressourcen (*Operating System Layer*). Nutzer können frei ihre eigenen virtuellen Computer-Cluster gestalten und sind somit auch für die Auswahl, die Installation und den Betrieb sowie das Funktionieren ihrer Software selbst verantwortlich. Die freie Cloud-Software Open-Stack fällt in diese Kategorie und enthält alle dazu notwendigen Komponenten.



# Vier Liefermodelle

## Private Cloud

Infrastruktur der Cloud „gehört“ nur einem Kunden.

## Community Cloud

Diese Cloud wird exklusiv für eine Community zur Verfügung gestellt.

## Public Cloud

Die Ressourcen dieser Cloud werden für die breite Öffentlichkeit vergeben.

## Hybrid Cloud

Dabei handelt es sich um eine Mischung aus einer Private, Community oder Public Cloud.

Weiterhin gibt es gemäß der offiziellen Beschreibung von Cloud Computing des NIST noch vier Liefermodelle (*Deployment Models*), die beschreiben, wer Zugriff auf die Cloud erhält, beziehungsweise von wo dieser erfolgt:

### Private Cloud

Die Infrastruktur der Cloud wird nur einem Kunden zur Verfügung gestellt oder gehört diesem.

### Community Cloud

Diese Cloud wird exklusiv für eine Community zur Verfügung gestellt.

### Public Cloud

Die Ressourcen dieser Cloud werden für die breite Öffentlichkeit vergeben.

### Hybrid Cloud

Hierbei handelt es sich um eine Mischung aus einer Private, Community oder einer Public Cloud.

Für Unternehmen kann die folgende Umsetzung einer Hybrid Cloud interessant sein: Alle Daten werden grundsätzlich in einer Private Cloud vorgehalten und verwaltet. Um Überkapazitäten zu vermeiden, wird diese jedoch nur für den Normalbetrieb ausgelegt. Unvermeidbare Lastspitzen werden bei Bedarf in eine Public Cloud ausgelagert, wobei die sensiven Daten im selbsverwalteten Bereich verbleiben.

# Fünf Eigenschaften

## On-demand Self-service

Nutzer kann die Eigenschaften der Systeme direkt ändern.

## Broad Network Access

Ressourcen sind über Standardmechanismen über Netzwerk verfügbar.

## Resource Pooling

Ressourcen des Anbieters werden für mehrere Kunden zusammengefasst.

## Rapid Elasticity

Ressourcen können erweitert oder freigegeben werden.

## Measured Service

Nutzung von Ressourcen kann gemessen, kontrolliert und ausgewertet werden.

Vor allem definiert das NIST fünf Anforderungen, denen eine Cloud genügen soll:

### On-demand Self-service

Ein Nutzer kann ohne weitere Interaktion mit einem Service Provider die Eigenschaften der Systeme ändern. Die Serviceorientierung (*... as a Service*) steht hier im Vordergrund.

### Broad Network Access

Die Ressourcen sind mit Hilfe von Standardmechanismen über ein Netzwerk verfügbar. Dies erlaubt auch eine weiträumige geografische Verteilung der Ressourcen.

### Resource Pooling

Die Ressourcen des Anbieters (Rechenleistung, Speicher, Storage, Netzwerk) werden in einem Pool zusammengefasst und können in einem Multi-Tenant-Modell mehreren Kunden zur Verfügung gestellt werden. Die physischen und virtuellen Ressourcen werden dem Kunden dabei dynamisch und ortsunabhängig je nach Bedarf zugeteilt. Die Virtualisierung spielt dabei eine wesentliche Rolle.

### Rapid Elasticity

Ressourcen können erweitert oder freigegeben werden. Dies kann unter Umständen auch automatisch geschehen, damit schnell steigende Anforderungen automatisch bedient werden können. Für den Nutzer erscheinen die Ressourcen unlimitiert. Ergebnis ist die gewünschte massive Skalierbarkeit.

## **Measured Service**

Die Nutzung von Ressourcen kann gemessen, kontrolliert, ausgewertet und für Anbieter und Nutzer transparent abgerechnet werden. Zusätzlich sollen Ressourcen durch das Cloud-System automatisch kontrolliert und optimiert werden.

OpenStack erfüllt alle diese Anforderungen.

### 1.3 Was ist OpenStack?



## Entstehungsgeschichte

- bildet als Open Source Cloud Computing Platform eine IaaS-Cloud ab
- mehrere Kernprojekte, die einzeln eingesetzt und einfach implementiert werden können
- Rackspace entwickelt 2005 die „Rackspace Cloud“, 2009 Entschluss zum Neuschreiben der Software
- 2010 Freigabe als Open Source Software
- NASA gibt im Mai 2010 „Nebula“ frei
- Juni 2010 Zusammenschluss der Projekte als OpenStack
- steht vollständig unter der Apache 2.0 Lizenz

Im Jahr 2010 von Rackspace sowie NASA gestartet, wird OpenStack als freie Softwarelösung für eine Infrastructure as a Service-Umgebung von mittlerweile über 200 Firmen aktiv entwickelt und unterstützt.

OpenStack bildet eine IaaS-Cloud ab. Dabei ist das Ziel des Projektes, frei interpretiert, die universelle Open Source Cloud Computing Platform für private und öffentliche Clouds zu werden.

Es existieren mehrere Kernprojekte, aus denen OpenStack besteht. Jedes davon kann einzeln eingesetzt und einfach implementiert werden.

Der gesamte Code des OpenStack-Projektes steht unter der Apache 2.0 Lizenz und ist über das Internet frei verfügbar.

2005 entwickelte die Firma Rackspace die Rackspace Cloud und entschloss sich 2009, die Software neu zu schreiben. Im März 2010 wurde der Storage-Teil dann als Open Source freigegeben. Im Mai 2010 gab die NASA ihr Nebula-Projekt frei und bereits im Juni schlossen sich die beiden Projekte zu OpenStack zusammen. Im Juli 2010 fand das erste Zusammentreffen der Entwickler statt und schon begann man mit den Veröffentlichungen der einzelnen Versionen.

Das Projekt steht vollständig unter der Apache 2.0 Lizenz.



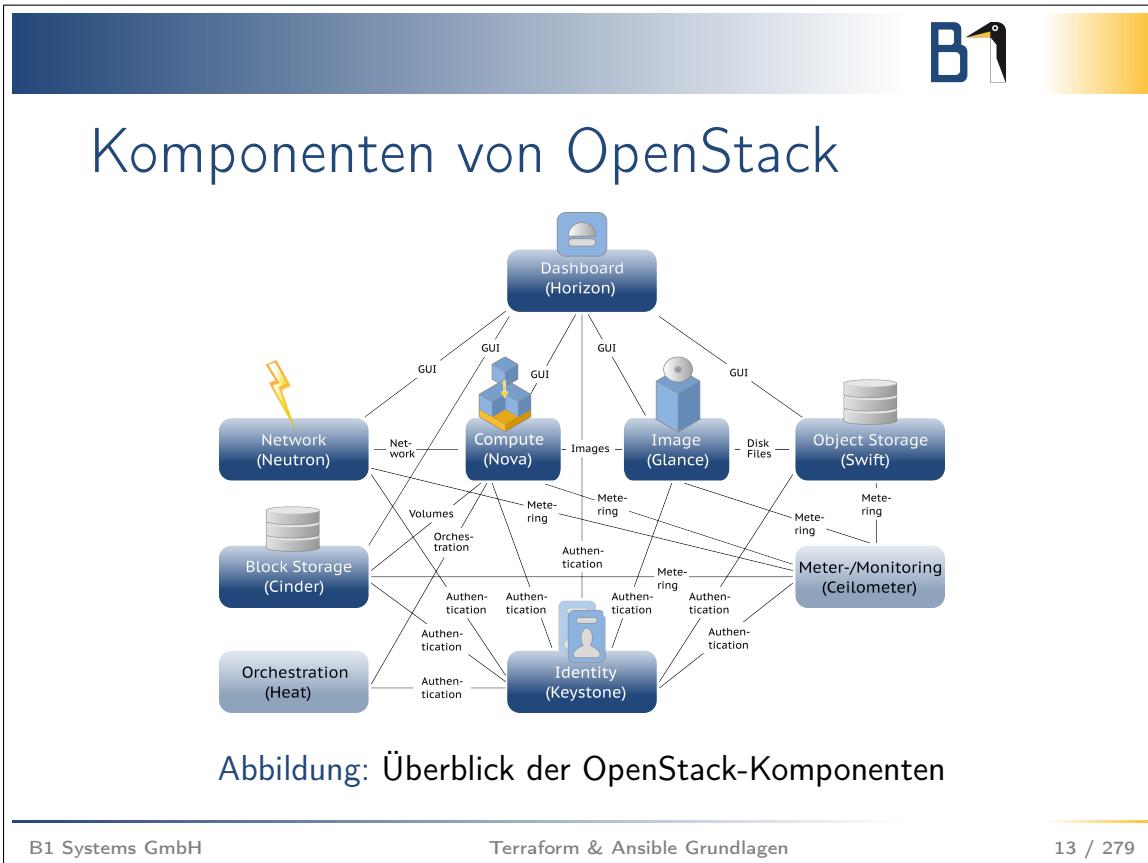
## Versionen

- Releasenummer: <Jahreszahl>.<Versionsnummer> (z.B. 2010.1)
- Codename meist Städte oder Counties in der Nähe der Design Summits, mehr:  
<http://wiki.openstack.org/ReleaseNaming>
- erste Version: 2010.1 („Austin“) im Oktober 2010
- aktuelles Release: 2014.2 („Juno“), Oktober 2014
- nächstes Relesase: 2015.1 („Kilo“), April 2015

OpenStack-Releases werden nummeriert mit Jahreszahl, gefolgt von einem Punkt und einer Versionsnummer, also z. B. „2014.2“ für das aktuelle „Juno“-Release, dem ersten Release im Jahr 2014. Die Codenamen waren bisher Städte und Counties in der Nähe des jeweils nächsten *Design Summits* von OpenStack, mit alphabetisch aufsteigenden Anfangsbuchstaben beginnend, angefangen mit „Austin“ im Oktober 2010. Das nächste Release, das im April 2015 erscheint, trägt den Codenamen „Kilo“.

VERSION	ERSCHEINUNGSMONAT
Austin	Oktober 2010
Bexar	Februar 2011
Cactus	April 2011
Diablo	September 2011
Essex	April 2012
Folsom	September 2012
Grizzly	April 2013
Havana	Oktober 2013
Icehouse	April 2014
Juno	Oktober 2014
Kilo	April 2015

## 1.3.1 Komponenten von OpenStack



OpenStack setzt sich aus einer Reihe einzelner Komponenten zusammen. Die Komponenten sind darauf abgestimmt, miteinander zu kommunizieren und zu arbeiten, können allesamt aber auch eigenständig betrieben werden. Die folgende Auflistung stellt die aktuellen Kernprojekte in der Reihenfolge ihrer Erscheinung vor:

### OpenStack Object Storage (Swift)

Swift bietet einen hochverfügbaren und verteilten Object Storage, ähnlich Amazons S3. Auf dem Storage können Templates und Snapshots virtueller Systeme abgelegt werden, aber auch – unabhängig von anderen OpenStack-Komponenten – Bilder, Videos oder Dokumente. So wird Swift beim Provider Rackspace zum Hosten der Cloud Files genutzt. Über diesen Dienst kann Speicherplatz in der Cloud genutzt werden. Die Daten können über eine API hinterlegt und verwaltet werden, jedoch kann der Storage nicht als Block Storage genutzt oder gemountet werden.

### OpenStack Compute (Nova)

Compute stellt die zentrale Komponente einer IaaS-Umgebung bereit. Hierüber werden Instanzen (virtuelle Maschinen) bereitgestellt, die auf einem Hypervisor (Xen oder KVM) betrieben werden.

### **OpenStack Image Service (Glance)**

Zur einheitlichen Bereitstellung von Templates und Snapshots virtueller Systeme werden Imagedateien für virtuelle Maschinen über eine zentrale Registrierung, den Image Service, verwaltet. Im Hintergrund können unterschiedliche Storage Backends genutzt werden. Glance bietet eine RESTful API, die auch das Abfragen von Metadaten der VM-Images unterstützt.

### **OpenStack Identity Service (Keystone)**

dient dem Identitätsmanagement und der Authentifizierung. Er übernimmt die zentrale Verwaltung von Kunden, Benutzern sowie Rollen und stellt einen Katalog aller in einer OpenStack-Umgebung vorhandenen Dienste bereit. Dadurch kann er als zentraler Einstiegspunkt in die Umgebung genutzt werden. Es können unterschiedliche Backends angebunden und genutzt werden, beispielsweise eine Datenbank oder ein LDAP-Server.

### **OpenStack Dashboard (Horizon)**

ist eine Weboberfläche zur zentralen Verwaltung der einzelnen OpenStack-Komponenten. Auf Basis eines frei erweiterbaren Django Frameworks soll in Zukunft darüber der komplette Funktionsumfang aller OpenStack-Komponenten bereitgestellt werden.

### **OpenStack Block Storage (Cinder)**

Eine eigenständige Komponente für das Bereitstellen von Block Storage in Form von Volumes für Cloud-Instanzen. Sie ersetzt den bisher in Nova integrierten Block Service `nova-volume`. Cinder-Volumes können einer virtuellen Instanz im laufenden Betrieb zugewiesen werden und so den dort laufenden Applikationen als Speicher dienen. Cinder kann dabei mit unterschiedlichen Storage Backends umgehen, beispielsweise mit iSCSI/LVM, NFS, NetApp oder einem Ceph-Cluster.

### **OpenStack Networking (Neutron)**

ist eine eigenständige Netzwerkkomponente, die über ein Plugin-System auf unterschiedliche Art und Weise Netzwerkfunktionalität bereitstellen kann. Auf Schicht 2 können wegen der modularen Struktur verschiedene Plugins wie Linux Bridge und Open vSwitch eingesetzt werden. Einzelne Agents kümmern sich etwa um die Vergabe von IP-Adressen und die Weiterleitung der Pakete auf Schicht 3. Die Neutron API erlaubt den Kunden eine unabhängige Netzwerkkonfiguration und erweiterte Kontrollmöglichkeiten wie Quality-of-Service, Monitoring sowie erweiterte Dienste wie Firewall, VPN und Intrusion Detection.

### **OpenStack Orchestration (Heat)**

ist ein Dienst, über den Cloudanwendungen orchestriert, d. h. automatisch auf viele Instanzen verteilt werden können. Heat bietet eine RESTful API an, die die AWS Cloudformation API implementiert. Dabei werden Vorlagen eingesetzt, die Beschreibungen der zu konfigurernden Dienste beinhalten.

## 1 Grundlagen des Cloud Computing

### **OpenStack Telemetry (Ceilometer)**

bietet Monitoring und Metering innerhalb einer OpenStack-Cloud-Struktur. Die gesammelten Daten können später ausgewertet werden, etwa für die Abrechnung der bereitgestellte Dienste.

Mit Icehouse akzeptierte Inkubatoren:

**Ironic** Bare-Metal-Deployment

**Marconi** Queue-Service

**Sahara** Data-Processing – Hadoop-Cluster

**Barbican** Key Management

## 2 Einführung in Terraform



# Einführung in Terraform

## 2.1 Zielsetzung



# Zielsetzung

## Dieses Kapitel

- stellt Terraform vor
- erklärt in diesem Zusammenhang das Konzept von *Infrastructure as Code* (IaC)
- wirft einen kurzen Blick auf die besonderen Vorteile und Features von Terraform
- zeigt Einsatzbereiche von Terraform
- weist auf Alternativen und verwandte Tools hin

## 2.2 Was ist Terraform?



# Was ist Terraform?

- Infrastructure Build Tool der Firma HashiCorp Inc.  
⇒ Infrastrukturmanagement für Cloud-Computing-Dienste  
(vs. Konfigurationsmanagement einzelner Rechner)
- Erstellen, Provisionieren und Verwalten von Infrastrukturen  
mittels Code-Schnipseln:  
⇒ *Infrastructure as Code* (IaC)
- Verwaltung mittels Kommandozeilenwerkzeug und  
Textdateien für Konfigurationen
- speichert den kompletten Lebenszyklus einer Anwendung  
(Versionierung wie bei Code in DevOps)
- Open Source, veröffentlicht unter der Mozilla Public  
License (2.0)
- in Go programmiert

Um den Anforderungen nach immer kürzer werdenden Entwicklungszyklen gerecht zu werden, besteht zunehmender Bedarf nach Automatisierung beim Aufbau einer Infrastruktur (z. B. zur Vorbereitung auf neue Software-Releases). Eine Möglichkeit, dies umzusetzen, liefert ein Konfigurationsmanagement, bei dem zunächst nur die Konfiguration beispielsweise einer virtuellen Maschine beschrieben wird. Mit Hilfe dieser Konfigurationsbeschreibung kann die virtuelle Maschine später dann provisioniert und je nach Bedarf weiter angepasst oder aktualisiert werden.

Terraform nutzt dieses Prinzip, um nicht nur einzelne Systeme, sondern komplettete Infrastrukturen – speziell in Cloud-Umgebungen – zu provisionieren und anzupassen. Terraform hilft dabei, gemäß einer Anleitung – einer Art „Bauplan“ – virtuelle Maschinen und andere Ressourcen einer geforderten Infrastruktur zu definieren und dann automatisiert bei einem der unterstützten Cloud-Anbieter auszurollen. Die Cloud kann im eigenen Rechenzentrum oder bei einem Anbieter von Cloud-Diensten (Provider) liegen und auch über mehrere Anbieter verteilt sein. Terraform ermöglicht dabei nicht nur ein reines Provisionieren einer Infrastruktur, sondern erlaubt es auch, bereits provisionierte Umgebungen nachträglich zu verändern und anzupassen.

Zur Konfiguration nutzt Terraform Textdateien, die die Dateiendung `.tf` tragen. Als Format für die Konfigurationsdateien haben Sie die Wahl zwischen *JavaScript Object Notation*

## 2 Einführung in Terraform

(JSON) und einer eigenen Konfigurationssprache, der JSON-ähnlichen *HashiCorp Configuration Language* (HCL); letztere wurde speziell für Terraform entwickelt, unterstützt Kommentare und bietet einige Erweiterungen, die die Code-Definition vereinfachen.

Ein Terraform-Projekt besteht aus Code-Blöcken, die jeweils genau eine Infrastrukturkomponente beschreiben, die in der Cloud erstellt werden soll. Jeder Code-Block wird mit einem Schlüsselwort eingeleitet, das die Art der Komponente definiert (z. B. provider, resource, variable, output, record, ...). Die häufigsten Code-Blöcke eines Terraform-Projekts sind dabei resource-Blöcke, die Infrastrukturelemente wie Rechnerinstanzen, virtuelle Netzwerke oder auch Security Groups oder DNS-Records beschreiben.

Vereinfacht gesagt ermöglicht es Terraform Administratoren und Entwicklern, mit HCL Dateien zu schreiben, die Definitionen ihrer gewünschten Ressourcen bei den gängigsten Providern (AWS, Azure, GCP, GitHub, Docker, ...) enthalten, und die Erstellung dieser Ressourcen zum Zeitpunkt der Anwendung zu automatisieren.



# Terraform – Features

**Infrastructure as Code** Blueprints einer Infrastruktur in Codeform

**Execution Plans** Planungsstufe für einen „Dry-Run“

**Kommandozeilenwerkzeug** CLI-Tool `terraform`

**HCL-Konfigurationsdateien** Konfiguration in Textdateien mit eigener Konfigurationssprache (HCL)

**Resource Graph** Graph aller Ressourcen zeigt die Abhängigkeiten einer Infrastruktur

**Plugins** offene Plugin-Architektur unterstützt unterschiedliche (Cloud-)Provider

**Multicloud Deployment** Infrastrukturen über mehrere Cloud-Provider hinweg

**Versionierung** wie in DevOps-Umgebungen (z. B. Git, SVN)

## Infrastructure as Code

Terraform beschreibt eine Infrastruktur mit einer High-Level-Konfigurationssprache, die eine Art Bauplan/Blueprint einer IT-Infrastruktur in Codeform erstellt. So kann eine Infrastruktur einfach geteilt, wieder verwendet oder verändert werden.

## Execution Plans

Terraform kennt eine Planungsstufe, bei der ein sog. „Execution Plan“ zeigt, was im Fall einer Anwendung ablaufen würde („Dry-Run“). Damit können unliebsame Überraschungen beim Ändern von Infrastrukturen vermieden werden.

## Kommandozeilenwerkzeug

Die Administration erfolgt über das Kommandozeilenwerkzeug `terraform`, mit dem sich sowohl die Planung („Dry-Run“) als auch die Ausführung steuern lässt.

## Konfigurationsdateien

Zur Konfiguration nutzt Terraform Textdateien, die die Dateiendung `.tf` tragen. Als Format für die Konfigurationsdateien haben Sie die Wahl zwischen *JavaScript Object Notation* (JSON) und – bevorzugterweise – einer eigenen Konfigurationssprache, der JSON-ähnlichen *HashiCorp Configuration Language* (HCL); letztere wurde speziell für Terraform entwickelt, unterstützt Kommentare und bietet einige Erweiterungen, die die Code-Definition für Konfigurationen vereinfachen.

## Resource Graph

Terraform erstellt einen Graph aller Ressourcen, den sog. „Resource Graph“, und

## 2 Einführung in Terraform

parallelisiert die Erstellung bzw. Veränderung aller nicht-abhängigen Ressourcen. Dadurch gestaltet sich der Infrastrukturaufbau so effizient wie möglich und ein Operator bekommt Einblick in die Abhängigkeiten der Infrastruktur. Den Graphen können Sie sich in Form einer DOT-Datei visualisieren lassen, die dann z. B. mit LibreOffice oder MS Office geöffnet und weiter verwendet werden kann.

### Plugins

Durch eine offen Plugin-Architektur können unterschiedliche (Cloud-)Provider unterstützt und neue Provider integriert werden. Derzeit werden über 70 Provider unterstützt.

### Multicloud Deployment

Eine Terraform-Infrastruktur kann auch über mehrere Cloud-Provider hinweg installiert werden, erlaubt also ein sog. „Multicloud Deployment“.

### Versionierung

Die Konfigurationsdateien lassen sich – analog wie bei Code in DevOps-Umgebungen – mit Tools wie Git oder SVN versionieren, sodass der komplette Lebenszyklus eines Terraform-Projekts aufgezeichnet wird. Terraform speichert die angelegte Infrastruktur und alle späteren Änderungen oder Erweiterungen, sodass alle Änderungen nachvollzogen und durch die Versionierung auch wieder einfach rückgängig gemacht werden können. Auch das Arbeiten im Team wird durch die Versionierung wesentlich erleichtert.

## 2.3 Infrastructure as Code (IaC)



### Was ist „Infrastructure as Code (IaC)“?

- *Infrastructure as Code (IaC) = (versionierter) Code als Bauplan einer Infrastruktur in Cloud-Umgebungen*
- stellt auf Basis von maschinenlesbarem Code definierte Cloud-Ressourcen bereit
- ⇒ Cloud-Automatisierung („Konfigurationsmanagement für Cloud-Orchestrierung“)
- liefert das Konfigurationsmanagement für das Cloud-Computing-Modell *Infrastructure as a Service (IaaS)*
- Infrastrukturelemente werden wie Software in Code-Schnipseln definiert („programmiert“ ) und verwaltet

Das Konzept, versionierten Code als Bauplan einer Infrastruktur vorzuhalten, wird als *Infrastructure as Code (IaC)* bezeichnet. IaC stellt IT-Infrastrukturleistungen wie Rechenleistung, Speicher und Netzwerk auf Basis maschinenlesbarer Definitionsdateien zur Verfügung. Die Infrastruktur wird dabei ähnlich wie Software „programmiert“, d. h. in maschinenlesbarem Code modelliert und immer wieder angepasst. IaC ist also der Prozess der Bereitstellung und Verwaltung von IT-Infrastrukturen mittels maschinenlesbarer Definitionsdateien – anstelle einer physischen Hardwarekonfiguration oder interaktiver Konfigurationstools. Definitionsdateien in einer Konfigurationssprache beschreiben eine geplante Zielinfrastruktur vollständig. Abhängigkeiten zwischen Teilen der Infrastruktur werden selbstständig ermittelt und die Ressourcen in korrekter Reihenfolge erstellt.

Eine manuelle Konfiguration physischer Hardwarekomponenten oder die mühsame Nutzung interaktiver, plattformspezifischer Konfigurationstools bleibt so erspart.

Das IaC-Konzept ist eng mit dem DevOps-Konzept und dem Cloud-Computing-Modell *Infrastructure as a Service (IaaS)* verknüpft. Das DevOps-Konzept vereint Software-Entwicklung („Development“) und Betrieb („Operations“) zu einer Einheit und schließt dabei auch die Systemadministration mit ein. Indem bei IaC die Infrastruktur wie Code behandelt wird, macht das Konzept die Administration den üblichen Werkzeugen und

## 2 Einführung in Terraform

Vorgehensweisen der Software-Entwicklung zugänglich, z. B. der Versionskontrolle. Das wiederum erleichtert die Verbindung von Development und Operations zu „DevOps“.

Die Anwendung von IaC stellt definierte Cloud-Ressourcen (d. h. IT-Infrastrukturleistungen wie Rechenleistung, Speicher und Netzwerk) auf Basis von maschinenlesbarem Code bereit und liefert somit das Konfigurationsmanagement für das Cloud-Computing-Modell *Infrastructure as a Service* (IaaS), in dem die Konfiguration der Ressourcen (virtuelle Maschinen, Container, Storage, Netzwerke, Security Groups, ...) in Code-Schnipseln definiert und diese wie Code versioniert werden.

Beim DevOps-Konzept werden Software-Entwicklung (*Development*) Hand-in-Hand mit Betrieb (*Operations*) verknüpft, einschließlich der dabei nötigen Systemadministration. IaC hilft dabei, mittels Definitionsdateien einer standardisierten Konfigurationssprache eine Zielaufbau vollständig zu beschreiben.

Im Cloud-Umfeld, das Virtualisierung wie KVM und Container-Architekturen wie Docker nutzt, ist der Begriff „Orchestrierung“ für den Prozess der Provisionierung, Steuerung und Kontrolle einer gewünschten Infrastruktur gebräuchlich, wohingegen im Bereich der klassischen Systemadministration der Begriff „Konfigurationsmanagement“-Software für Orchestrierungstools der Cloud-Generation üblich ist.

Hierzu zählen beispielsweise Ansible, Chef, Salt und Puppet, die sich für IaC einsetzen lassen. Daran zeigt sich, dass dieses Konzept nicht erst mit Cloud und DevOps entstanden ist, sondern seine Wurzeln in der Systemadministration hat.

Weitere Informationen finden Sie unter:

<https://learn.hashicorp.com/tutorials/terraform/infrastructure-as-code>

## 2.4 Provider



# Provider

Mögliche Services, mit denen Terraform genutzt werden kann:

- *Infrastructure as a Service* (IaaS)
- *Platform as a Service* (PaaS)
- *Software as a Service* (SaaS)

Provider-Plugins (Auswahl):

- Apache CloudStack
- Amazon Web Services (AWS)
- Azure
- GitHub
- Google Cloud Platform (GCP)
- OpenStack
- OpenTelekomCloud
- VMware

Terraform ermöglicht es, Infrastruktur in Code abzubilden und diesen Code auf verschiedene Anbieter, sogenannte *Provider*, zur Definition und Verwaltung einer gewünschten IT-Infrastruktur anzuwenden. Da Terraform anbieteragnostisch und durch eine offene Plugin-Architektur erweiterbar ist, können Sie die Plattformen unterschiedlicher Cloud-Dienstleister und Service-Anbieter nutzen, um dort Terraform-Infrastruktur-Code einzuspielen und damit bedarfsgerechte Infrastruktur-Umgebungen aufzubauen.

Terraform kann auf mehreren Service-Leveln eingesetzt werden, nicht nur zur Einrichtung von IaaS (*Infrastructure as a Service*)-Umgebungen, sondern auch in PaaS (*Platform as a Service*)- und SaaS (*Software as a Service*)-Umgebungen.

Terraform-Plugins gibt es derzeit für über 70 Cloud-Provider, die Dienste wie IaaS, SaaS und PaaS anbieten, u. a.:

- Apache CloudStack
- Amazon Web Services (AWS)
- Azure
- BaiduCloud
- Brightbox

## 2 Einführung in Terraform

- DigitalOcean
- GitHub
- Google Cloud Platform (GCP)
- Hetzner Cloud
- HuaweiCloud / HuaweiCloudStack
- OpenNebula
- OpenStack
- OpenTelekomCloud
- OVH
- TelefonicaOpenCloud
- VMware
- Vultr
- Yandex.Cloud
- 1&1

Eine aktuelle Liste wird auf `terraform.io` unter folgender Adresse gepflegt:

<https://www.terraform.io/docs/providers/index.html>

So können Sie z. B. eine sauber konfigurierte Versionskontrolle für Ihr Projekt auf GitHub einrichten und für einen LAMP-Stack als Plattform zum Erstellen der dazu nötigen Infrastruktur die Amazon Web Services (AWS) nutzen.

## 2.5 Einsatzbereiche



# Einsatzbereiche

Bereitstellen von IT-Infrastrukturen mit:

- virtuellen Maschinen (KVM, OpenStack, VMware, ...)
- Containern (z. B. Docker, Kubernetes)
- weiteren Cloud-Ressourcen (Netzwerke, Storage, ...)

z. B. für:

- Entwicklungs- und Testumgebungen
- Multicloud Deployment
- Konfigurationsanpassungen (z. B. DNS- oder Monitoring-Server)
- Scaleout

Terraform kann nahezu alle Arbeiten erledigen, die beim Deployment von virtuellen Infrastrukturen anfallen. Dazu gehört das Bereitstellen von IT-Infrastrukturen mit

- virtuellen Maschinen (mit KVM, OpenStack, VMware, ...)
- Containern (z. B. Docker, Kubernetes)
- weiteren Cloud-Ressourcen (Netzwerke, Storage, ...)

z. B. für:

- Entwicklungs- und Testumgebungen (z. B. Kerneltests wie Kdevops auf Vagrant, Terraform und Ansible)
- Multicloud Deployment
- Konfigurationsanpassungen (z. B. DNS- oder Monitoring-Server)
- Erweiterung bestehender Infrastrukturen („Scaleout“)

Der Aufbau einer solchen IT-Infrastruktur kann im eigenen Rechenzentrum, bei Cloud-Providern oder auch über mehrere Cloud-Provider hinweg erfolgen.

## 2.6 Alternativlos?



# Alternativlos?

- Ansible
- Chef Infra
- Jenkins
- Kubernetes
- Packer
- Cloud Foundry
- Serverless
- Pulumi
- ... (more to come?)

Auch Terraform ist nicht alternativlos, findet aber aufgrund seiner besonderen Fähigkeiten in Cloud-Umgebungen in den letzten Jahren zunehmend Anwender.

Andere bekannte Software-Lösungen bzw. Management-Frameworks mit Fähigkeiten für IaC-Anwendungen sind (Auswahl):

### Ansible

Ansible ist ein Open-Source Automatisierungs-Werkzeug von *AnsibleWorks, Inc.* zur Orchestrierung, allgemeinen Konfiguration und Administration von Rechnern. Es kombiniert Softwareverteilung, Ad-hoc-Kommando-Ausführung und Konfigurationsmanagement. Die Verwaltung von Netzwerkcomputern erfolgt unter anderem über SSH und erfordert keinerlei zusätzliche Software auf dem zu verwaltenden System. Module nutzen zur Ausgabe JSON und können in jeder beliebigen Programmiersprache geschrieben sein. Das System nutzt YAML zur Formulierung wiederverwendbarer Beschreibungen von Systemen. Ansible beherrscht durch eine *Multi-Node Orchestration Engine* ebenfalls Orchestrierung, kennt aber weniger Cloud-spezifische Verwaltungsfunktionen.

<http://www.ansible.com/>

### Chef Infra

Chef ist wie Ansible ein altbekanntes Konfigurationsmanagement-Tool und ver-

wendet eine reine Ruby-Domain-spezifische Sprache zum Schreiben sog. „Rezepte“ für die Systemkonfiguration. Mit *Chef Infra* wurde eine speziell auf Cloud-Architekturen ausgerichtete Version von Chef entwickelt.

<https://www.chef.io>

### Kubernetes

Kubernetes ist ein Container-orientiertes Managementframework, das Containerbasierte Anwendungen automatisiert provisionieren und verwalten kann und so die Vorteile von IaaS und PaaS vereint. Es kann neben Rechenressourcen auch z. B. Storage, Netzwerke, HA, uvm. bereit stellen. Kubernetes wurde ursprünglich von Google entworfen und später an die Cloud Native Computing Foundation (CNCF) gespendet.

<https://kubernetes.io/>

### Packer

Wie Terraform von HashiCorp, legt Packer den Schwerpunkt auf die Erstellung sog. „Golden Images“ aus einer bestimmten Grundkonfiguration für unterschiedliche Plattformen (z. B. Clouds).

<https://www.packer.io>

### Cloud Foundry

OpenSource-Multi-Cloud-Anwendungsplattform als Service (= Ausrichtung auf *Platform as a Service* (PaaS)); wird von der Cloud Foundry Foundation, einer 501-Organisation, verwaltet.

<https://www.cloudfoundry.org>

### Pulumi

Der Hauptmitbewerber von Terraform, im Funktionsumfang ziemlich Terraform-ähnlich, nutzt aber – im Gegensatz zu Terraform – bekannte Sprachen.

<https://www.pulumi.com>

### Jenkins

Ursprünglich als erweiterbares, Web-basiertes Software-System zur kontinuierlichen Integration (*Continuous Integration, CI*) von Software-Komponenten entwickelt, kann Jenkins prinzipiell zur Verwaltung von Konfigurationen in Textdateien verwendet werden. Das Einrichten von Jenkins ist jedoch komplex und es fehlen viele Cloud-spezifischen Infrastrukturanpassungen.

<https://jenkins.io/>

### Serverless

Framework speziell zur Nutzung von AWS-Infrastruktur-Ressourcen; flexibel in der Sprache (möglich sind z. B. Node.js, Python, Java), weniger flexibel in der Nutzung des Providers; kann mit Terraform kombiniert werden.

<https://www.serverless.com>

## 2.7 Terraform und Vagrant



# Terraform und Vagrant

- beides Projekte von HashiCorp Inc.
- Vagrant Werkzeug zum Einrichten von Entwicklungsumgebungen (Erstellen und Verwalten von VMs, Softwareverteilung)
  - ⇒ für kleine(re) Entwicklungsumgebungen
- Terraform Einrichtung von (auch sehr großen, verteilten) Infrastrukturen
  - ⇒ hauptsächlich zum Verwalten von Remote-Ressourcen bei Providern mit speziellen Features wie:
    - Synchronisieren von Ordnern
    - automatisiertes Networking
    - HTTP-Tunnelung
- Kombination von Vagrant und Terraform möglich

*Vagrant* (<https://www.vagrantup.com/>) ist ein häufig genutztes Tool zum Erstellen und Verwalten virtueller Maschinen. Vagrant fungiert dabei als Wrapper zwischen einer Virtualisierungssoftware wie KVM/QEMU, VirtualBox, VMware oder Hyper-V und einem Konfigurationsmanagement wie Ansible, SaltStack, Puppet – oder auch Terraform: Vagrant kann mit Terraform kombiniert werden.

Sowohl Vagrant als auch Terraform sind Projekte der HashiCorp Inc., allerdings mit unterschiedlichem Fokus: Vagrant ist ein Werkzeug zum Einrichten von kleine(re)n Entwicklungsumgebungen mit wenigen VMs, seine Schwerpunkte liegen vor allem bei der Softwareverteilung („Deployment“; insbesondere bei der Software- und Web-Entwicklung). Vagrant eignet sich vor allem für kleine(re) (lokale) Entwicklungsumgebungen mit wenigen VMs, wohingegen sich Terraform mehr auf die Einrichtung von – auch sehr großen, verteilten – Infrastrukturen bei (Cloud-)Providern konzentriert, die auch über mehrere Clouds verteilt sein können („Multicloud Deployment“). Zum Verwalten dieser Remote-Ressourcen bei Providern wartet Terraform mit speziellen Features auf, z. B.:

- Synchronisieren von Ordnern
- automatisiertes Networking
- HTTP-Tunnelung

## 2.8 Terraform Enterprise



# Terraform Enterprise

- Terraform Enterprise = Terraform Cloud als Hosted Service
- private Terraform Cloud für Unternehmen mit Support und erweiterten Features:
  - Audit Logging
  - SAML Single Sign-on
  - keine Ressource Limits
- Adresse: <https://app.terraform.io>

B1 Systems GmbH      Terraform & Ansible Grundlagen      23 / 279

HashiCorp Inc. bietet in Form des sog. „Terraform Enterprise“ eine eigene Distribution der Terraform Cloud als Hosted Service an (<https://app.terraform.io>). Terraform Enterprise bietet Unternehmen eine private Terraform Cloud mit Support und erweiterten Features wie z. B.:

- Audit Logging
- SAML Single Sign-on

Terraform Enterprise kennt keine Ressource Limits – gibt jedenfalls keine bekannt.

Mehr zu Terraform Enterprise unter der Adresse <https://app.terraform.io>.

## 2.9 Links und Quellen



# Links und Quellen

- HashiCorp Inc.  
<https://www.hashicorp.com/>
- Terraform Tutorial  
<https://learn.hashicorp.com/tutorials/terraform/>
- Aktuelle Provider-Liste  
<https://www.terraform.io/docs/providers/index.html>
- Terraform Cloud als Hosted Service  
<https://app.terraform.io>.
- Vergleich verschiedener Terraform-Alternativen  
<https://stackshare.io/terraform/alternatives>

Nachfolgend als Quellenangabe und zur weiteren Information ein paar Links:

HashiCorp Inc.:

<https://www.hashicorp.com/>

Terraform Tutorial:

<https://learn.hashicorp.com/tutorials/terraform/>

Aktuelle Provider-Liste:

<https://www.terraform.io/docs/providers/index.html>

Terraform Cloud als Hosted Service:

<https://app.terraform.io>.

Vergleich verschiedener Terraform-Alternativen:

<https://stackshare.io/terraform/alternatives>

Artikel „Infrastruktur verwalten mit Terraform“:

<https://www.linux-magazin.de/ausgaben/2017/12/terraform/>

## 3 Installation von Terraform



# Installation von Terraform

### 3.1 Zielsetzung



## Zielsetzung

Dieses Kapitel zeigt die Installation von Terraform.

### 3.2 Terraform installieren



## Terraform installieren

- Download von <https://www.terraform.io>
- entpacken
- dem Pfad hinzufügen

Es empfiehlt sich immer, die neueste Version von Terraform zu verwenden. Die meisten Distributionen bieten es noch nicht oder in veralteten Versionen an. Generell sollten Sie eine Version  $\geq 0.13$  wählen.

Terraform bietet im Vergleich zu anderer OpenSource-Software keinen Support an. Je-doch gibt es die Möglichkeit, gegen Bezahlung mehr Funktionalität zu erhalten. Dazu gehört unter anderem eine Cloud-Umgebung von HashiCorp, die das Arbeiten in Teams erleichtert, Rollen und Benutzermanagement bietet, Job-Scheduling ermöglicht und API-Schnittstellen anbietet. Bei entsprechendem Bedarf ist es möglich, diese Cloud-Umgebung dediziert selbst zu hosten. Dies wäre für Audit-relevante Geschäftsbereiche von besonderem Interesse.

Das Binary kann hier heruntergeladen werden:

<https://www.terraform.io/downloads.html>

Da Terraform in Go programmiert wurde, entfällt ein distributionsspezifisches Binary. Abhängigkeiten zu anderen Bibliotheken entfallen ebenfalls.

### 3 Installation von Terraform

Derzeit ist Terraform für folgende Betriebssystem-Familien verfügbar:

- macOS
- FreeBSD
- Linux
- OpenBSD
- Solaris
- Windows

### 3.3 Aufgabenteil



## Aufgabenteil – Installation von Terraform

- ① Laden Sie Terraform herunter und entpacken Sie die heruntergeladene Datei.
- ② Verschieben Sie das Binary im Anschluss z. B. nach /usr/local/bin/terraform.

1. Laden Sie Terraform herunter und entpacken Sie die heruntergeladene Datei.
2. Verschieben Sie das Binary im Anschluss z. B. nach /usr/local/bin/terraform.

## Musterlösung

**1. Laden Sie Terraform herunter und entpacken Sie die heruntergeladene Datei:**

Laden Sie Terraform für Linux 64-Bit von  
<https://www.terraform.io/downloads.html>  
herunter. Die Datei anschließend mit

```
$ unzip terraform*.zip
```

entpacken.

**2. Verschieben Sie das Binary im Anschluss z. B. nach  
`/usr/local/bin/terraform`:**

```
$ sudo mv terraform /usr/local/bin/
```

## 4 Terraform CLI – Grundlegende Befehle



# Terraform CLI – Grundlegende Befehle

## 4.1 Zielsetzung



# Zielsetzung

## Dieses Kapitel

- liefert einen Überblick über die grundlegenden Befehle des Terraform CLI
- zeigt die Initialisierung eines Workspace

## 4.2 Terraform CLI



# Terraform CLI

### Die wichtigsten CLI-Befehle

```
$ terraform apply
$ terraform plan
$ terraform destroy
$ terraform fmt
$ terraform graph
$ terraform init
$ terraform show
$ terraform taint
```

---

B1 Systems GmbH      Terraform & Ansible Grundlagen      31 / 279

Die wichtigsten CLI-Befehle im Überblick:

**terraform apply** Infrastruktur aufbauen. Ein `apply` weist Terraform an, alle relevanten Dateien im aktuellen Verzeichnis einzulesen, syntaktisch zu überprüfen und dann die Anweisungen (nach Benutzerbestätigung) auszuführen. Terraform wird hierbei intern eine eigene Struktur aufbauen, in welcher Reihenfolge die Anweisungen abgearbeitet werden sollen. Somit können viele Aufgaben parallelisiert und so schneller bearbeitet werden. Zusätzlich wird noch die `terraform.tfstate` gelesen und verarbeitet. Sollten hier bereits Ressourcen existieren, so wird vor der eigentlichen Aktivität ein „Refresh“ zur Validierung dieser durchgeführt.

**terraform plan** Infrastruktur planen. Mit `plan` werden die selben Schritte wie bei einem `apply` ausgeführt, ohne dass tatsächlich etwas verändert wird. Dies gleicht einem „Dry-Run“. Es wird auch keine Änderung in der `terraform.tfstate` vorgenommen. Ein `plan` dient lediglich zur Visualisierung der Schritte, die bei einem `apply` ausgeführt würden.

**terraform destroy** Infrastruktur „abreißen“. Als Gegenteil zum `apply` entfernt ein `destroy` die Ressourcen wieder. Allerdings orientiert sich ein `destroy` nur am aus der `terraform.tfstate` gelesenen *State*.

**terraform fmt** Code automatisch formatieren. Bei größeren Code-Dateien können sich leicht Einrückungs- und Formatfehler einschleichen. Diese stören den Ablauf nicht, erschweren aber die Wartbarkeit. Um dem entgegenzuwirken, liefert Terraform eine eingebaute Funktion zur Reformatierung des Codes mit.

**terraform graph** Ausgabe in eine DOT-Datei zur Visualisierung. Diese Datei kann mit unterschiedlichen Programmen als eine UML-ähnliche Grafik angezeigt werden, um die Reihenfolge der Arbeitsschritte von Terraform zu visualisieren. Parallelе Arbeitsschritte sind immer auf der selben Horizontalachse abgebildet. Hiermit können ggf. logische Fehler debuggt werden. Je mehr Ressourcen definiert werden, desto unübersichtlicher wird die Darstellung.

**terraform init** Arbeitsverzeichnis initialisieren. Ein `init` wird mindestens einmal vor Anwendung von produktiven Anweisungen benötigt. Dazu zählen z. B. `apply` oder `destroy`, nicht aber `fmt` oder `show`. `init` weist Terraform an, benötigte Provider herunterzuladen oder einzubinden, Module zu laden und ggf. Backends zu verknüpfen. Weiteres zu Modulen, Backends, etc. entnehmen Sie bitte späteren Kapiteln.

**terraform show** Ressourcen aus aktuellem State anzeigen. Liest die Datei `terraform.tfstate` ein und gibt ihren Inhalt geordnet aus. Der Befehl lässt ebenfalls weitere Filterungen auf einzelne Ressourcen zu. Besonders hilfreich wird dies, wenn in der eigenen Tool-Chain auf Werte aus Terraform zugegriffen werden muss.

**terraform taint** Ressource als „defekt“ markieren, damit diese beim nächsten Aufruf von `terraform apply` neu erzeugt wird. Ein `taint` markiert lediglich eine Ressource für eine Neuerzeugung. Es wird keine Ressource gelöscht oder modifiziert. Erst beim nächsten `apply` wird auf die markierte Ressource erst ein `destroy`, danach ein `apply` ausgeführt. Ein typisches Anwendungsbeispiel wäre z. B. der Wechsel eines Image einer VM.

Eine vollständige Liste aller verfügbaren Kommandos ist hier zu finden:

<https://www.terraform.io/docs/commands/index.html>

## 4.3 Workspace anlegen



# Workspace anlegen

- Terraform arbeitet Ordner-basiert
- alle Dateien in einem Verzeichnis werden verarbeitet
- Unterordner werden *nicht* beachtet
- seit Terraform 0.13 wird ein *Terraform Settings Block* benötigt

Beispiel – Aktuelles Verzeichnis zu Arbeitsverzeichnis machen

```
$ mkdir /home/tux/terraform_project
$ cd /home/tux/terraform_project/
$ vi versions.tf
$ terraform init
```

Terraform arbeitet Ordner-basiert, d. h. alle Dateien im aktuellen Verzeichnis, die dem Namensschema entsprechen (z. B. `*.tf`, `*.auto.tfvars`; genauereres dazu im Kapitel zu Variablen), werden verarbeitet. Unterverzeichnisse werden *nicht* beachtet. Üblicherweise wird Terraform als normaler Benutzer gestartet und Projekte in nicht-Systemrelevanten Bereichen gespeichert (z. B. Home-Verzeichnis oder `/opt/`).

Mit `terraform init` wird Terraform angewiesen, das aktuelle Verzeichnis zu einem Arbeitsverzeichnis (*Workspace*) zu machen. Seit Version 0.13 muss hier jedoch mindestens eine Datei mit einem *Terraform Settings Block* vorhanden sein, mehr dazu auf der folgenden Seite. Bei der Initialisierung wird automatisch ein Verzeichnis `.terraform` angelegt, welches benötigte Komponenten für Terraform beinhaltet. Dies könnte z. B. ein Provider sein (siehe nächstes Kapitel). Ein `terraform init` wird mindestens einmal benötigt, bevor beispielsweise ein `terraform apply` erfolgen kann. Die Code-Dateien können schon vor dem `init` angelegt werden. Bei nachträglicher Ergänzung von Modulen oder weiteren Providern wird ebenfalls wieder ein `terraform init` notwendig.

## 4 Terraform CLI – Grundlegende Befehle

Im folgenden Beispiel wird beispielsweise der Provider für AWS automatisch heruntergeladen:

```
$ ~/terraform_project> terraform init

Initializing the backend...

Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "aws" (hashicorp/aws) 2.70.0...

The following providers do not have any version constraints in
    ↪ configuration,
so the latest version was installed.

To prevent automatic upgrades to new major versions that may contain
    ↪ breaking
changes, it is recommended to add version = "..." constraints to the
corresponding provider blocks in configuration, with the constraint
    ↪ strings
suggested below.

* provider.aws: version = "~> 2.70"

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan"
    ↪ to see
any changes that are required for your infrastructure. All Terraform
    ↪ commands
should now work.

If you ever set or change modules or backend configuration for
    ↪ Terraform,
rerun this command to reinitialize your working directory. If you
    ↪ forget, other
commands will detect it and remind you to do so if necessary.
```

## 4.4 Terraform Settings Block



# Terraform Settings Block

## Beispiel – Terraform Settings Block

```
terraform {
  required_providers {
    openstack = {
      source = "terraform-provider-openstack/openstack"
    }
  }
  required_version = ">= 0.13"
}
```

Dieses Beispiel zeigt einen *Terraform Settings Block*. Er enthält unter anderem Informationen zur vorausgesetzten Terraform-Version und zu den zu ladenden Providern. Aufgrund der neuen Möglichkeit, die Provider von mehreren Anbietern zu beziehen, ist die Provider-Angabe seit Version 0.13 von Terraform notwendig. Durch die Angabe von `source` wird nun deutlich, von welchem Anbieter der Provider geladen werden soll.

Üblicherweise ist dieser Code in der Datei `versions.tf` abgelegt, dies ist jedoch nicht verpflichtend. In Version 0.13 war es möglich, sich den Code für einen *Terraform Settings Block* generieren zu lassen, solange bereits weitere Terraform-Code-Schnipsel im selben Verzeichnis vorhanden waren:

```
$ terraform 0.13upgrade .
```

Dieses Kommando ist in Version 0.14 bereits wieder entfernt worden und nicht mehr vorhanden. Die Datei samt Inhalt muss jetzt händisch angelegt werden.



## 4.5 Aufgabenteil



### Aufgabenteil – Workspace initialisieren

Erstellen Sie einen Workspace  
`/home/<Benutzer>/terraform_project/`. Fügen Sie dem Verzeichnis eine Datei (z. B. `versions.tf`) mit dem *Terraform Configuration Block* hinzu.

Erstellen Sie einen Workspace `/home/<Benutzer>/terraform_project/`. Fügen Sie dem Verzeichnis eine Datei (z. B. `versions.tf`) mit dem *Terraform Configuration Block* hinzu.

## Musterlösung

**Erstellen Sie einen Workspace `/home/<Benutzer>/terraform_project/`. Fügen Sie dem Verzeichnis eine Datei (z. B. `versions.tf`) mit dem *Terraform Configuration Block* hinzu:**

```
$ mkdir /home/tux/terraform_project/
$ cd /home/tux/terraform_project/
$ vi versions.tf

terraform {
  required_providers {
    openstack = {
      source = "terraform-provider-openstack/openstack"
    }
    template = {
      source = "hashicorp/template"
    }
  }
  required_version = ">= 0.13"
}

$ terraform init
Terraform initialized in an empty directory!

The directory has no Terraform configuration files. You may begin
  ↪ working
with Terraform immediately by creating Terraform configuration files.
```

## 5 Terraform – Provider



# Terraform – Provider

## 5.1 Zielsetzung



# Zielsetzung

Dieses Kapitel

- liefert einen Überblick über die Funktionen eines Providers
- zeigt Code-Beispiele für Provider
- erklärt, warum Provider nicht direkt im Code notwendig sind

## 5.2 Funktionen eines Providers



# Funktionen eines Providers

- Plugins für Terraform, die API-Schnittstellen kennen
- zwei Aufgaben eines Providers:
  - Data Sources
  - Resources

## Provider – Konfiguration

```
provider "name" {
  parameter1 = "wert1"
  parameter2 = "wert2"
  ...
}
```

Provider sind Plugins für Terraform, welche die API-Schnittstellen eines Dienstes (z. B. AWS, *Amazon Web Services*) kennen. Die Kommunikation mit dieser API-Schnittstelle kann z. B. über HTTP, SSH oder andere Protokolle stattfinden. Dies ist vom entsprechenden Provider abhängig.

Allgemein kann man die Funktionen eines Providers in zwei Kategorien aufteilen.

*Data Sources* bieten die Möglichkeit, während der Laufzeit Daten zu ermitteln. Dies ist z. B. nützlich, um die ID eines bereits existenten Netzwerks zu erhalten, von dem nur der Name bekannt ist.

*Resources* sind für das Generieren von Ressourcen zuständig. In ihnen wird ein gewünschter Zustand beschrieben.

Grob können diese Kategorien also mit Get-Methoden und Set-Methoden aus der Programmierung verglichen werden.

Die Konfiguration der Provider erfolgt über Einträge in den `.tf`-Dateien. Dabei wird ein Konfigurationsblock immer durch das Stichwort „`provider`“ eingeleitet, gefolgt von einem eindeutigen internen Namen. In geschweiften Klammern folgen dann die gewünschten Parameter. Welche möglich bzw. notwendig sind, ist vom Provider-Typ abhängig.

### 5.3 Beispiel – OpenStack



## Beispiel – OpenStack

### Beispiel – OpenStack Provider

```
provider "openstack" {  
    user_name      = "tux"  
    tenant_name   = "tux-project"  
    password       = "f1c25939-c09a-4d3f-8eeb-83a7"  
    auth_url      = "https://my-cloud:5000/v3.0"  
    region        = "South-2"  
}
```

Das Beispiel zeigt die Definition eines Providers für OpenStack. Dieser Code wird (sofern er notwendig ist) wie auch alle anderen Code-Schnipsel in eine beliebige `*.tf`-Datei geschrieben.

Jeder Provider erhält über die Konfiguration einen eindeutigen, lokal gültigen Namen, unter dem er in der weiteren Konfiguration angesprochen wird. Dieser folgt auf das Stichwort `provider` – im Beispiel also `openstack`. Welche weiteren Parameter erwartet werden oder möglich sind, ist abhängig vom jeweiligen Provider-Typ.

Im Beispiel erfolgt der Zugriff auf OpenStack mittels des Providers unter Angabe von Benutzername (`user_name`) und Passwort (`password`). Je nach Provider ist beispielsweise auch die Angabe von API-Benutzer und API-Token möglich. Die `auth_url` legt die URL zum Zugriff auf den entsprechenden Dienst fest. Der `tenant_name` ist eine spezifische Angabe für OpenStack und legt den Tenant für die Anmeldung fest. Mit `region` geben Sie in einer OpenStack-Umgebung die passende Umgebung an.

## 5.4 Beispiel – AWS



### Beispiel – AWS

#### Beispiel – AWS Provider

```
provider "aws" {  
    region      = "us-west-2"  
    access_key  = "2e67b9d7-3d36-4230-acac-126f"  
    secret_key  = "0c501b68-4162-4db2-ab1a-e0b6"  
}
```

Das Beispiel zeigt eine mögliche Definition für einen AWS-Provider (*AWS – Amazon Web Services*). In diesem Beispiel ist der interne „Name“ des Providers `aws`. Beim Zugriff erfolgt die Authentifizierung über den angegebenen `access_key` und den passenden `secret_key`. Die `region` legt fest, auf welchen Bereich (Region) der *Amazon Web Services* zugegriffen werden soll.

## 5.5 Beispiel – GCP



### Beispiel – GCP

#### Beispiel – GCP Provider

```
provider "google" {  
    credentials = file("account.json")  
    project     = "my-project-id"  
    region      = "us-central1"  
}
```

Dieses Beispiel zeigt eine mögliche Konfiguration für einen GCP Provider (*GCP – Google Cloud Platform*). Dieser erhält in der Beispiel-Konfiguration den Namen `google`. Die Zugangsdaten zur Authentifizierung werden aus einer Datei im JSON-Format gelesen (`credentials = file(...)`). Weitere Angaben sind der Name des passenden Projekts (`project`) und die Region (`region`) der Google Cloud Platform.

## 5.6 Beispiel – Azure



### Beispiel – Azure

#### Beispiel – Azure Provider

```
provider "azure" {
    subscription_id = "56e13f3e-44d4-4670-be12-1e7b"
    certificate      = file("my_cert.crt")
}
```

Das Beispiel zeigt eine mögliche Konfiguration für einen Azure Provider. Dabei werden die `subscription_id` und der Name zu einer Zertifikats-Datei (`certificate`) übergeben.

## 5.7 Weitere Provider



# Weitere Provider

- Null
- Local
- HTTP
- KVM (nicht offiziell)

Eine vollständige Auflistung aller offiziellen Provider ist hier zu finden:  
<https://registry.terraform.io/browse/providers>

Einige weitere mögliche Provider:

**Null** Vorrangig für Aktionen, die sich nicht durch andere Provider abbilden lassen; häufig in Kombination mit *Provisionern* genutzt um z. B. während der Laufzeit einen SSH-Befehl abzusetzen.

**Local** Für das Managen / Einlesen von lokalen Objekten wie z. B. Dateien; wird z. B. häufig verwendet, um `cloud-init` Dateien einzulesen.

**HTTP** Sammelt via HTTP GET-Daten von einer URL und stellt diese bereit; das Senden von Daten per HTTP PUT oder HTTP POST ist *nicht* möglich.

**KVM** Ist ein Provider für KVM / libvirt Virtualisierung, der auf GitHub frei verfügbar ist; keine offizielle Unterstützung seitens HashiCorp.

## 5.8 Arbeiten ohne Provider im Code



# Arbeiten ohne Provider im Code

- Provider müssen nicht im Code definiert werden
- Umgebungsvariablen müssen gesetzt sein
- werden automatisch erkannt

### Beispiel – Umgebungsvariablen für OpenStack Provider

```
$ export OS_AUTH_URL=https://api-1.betacloud.de:5000/v3
$ export OS_PROJECT_ID=9f416c2718284525b1b37315cfadba9f
$ export OS_PROJECT_NAME="training-terraform"
$ export OS_USER_DOMAIN_NAME="b1systems"
$ export OS_PROJECT_DOMAIN_ID="7ace1917796541d2a5660dba84d3f664"
$ export OS_USERNAME="tux"
$ export OS_PASSWORD=linux
$ export OS_REGION_NAME="betacloud-1"
$ export OS_INTERFACE=public
$ export OS_IDENTITY_API_VERSION=3
```

Eine Provider-Definition im Code kann auch vollständig weggelassen werden, wenn unterstützt. Dazu müssen bestimmte Umgebungsvariablen gesetzt sein. Terraform erkennt dann anhand der verwendeten Ressourcen, welche Provider notwendig sind. Die Provider selber lesen die Werte der Umgebungsvariablen ein und verwenden diese.

Hier ein Beispiel der notwendigen Variablen für einen OpenStack Provider:

```
$ export OS_AUTH_URL=https://api-1.betacloud.de:5000/v3
$ export OS_PROJECT_ID=9f416c2718284525b1b37315cfadba9f
$ export OS_PROJECT_NAME="training-terraform"
$ export OS_USER_DOMAIN_NAME="b1systems"
$ export OS_PROJECT_DOMAIN_ID="7ace1917796541d2a5660dba84d3f664"
$ export OS_USERNAME="tux"
$ export OS_PASSWORD=linux
$ export OS_REGION_NAME="betacloud-1"
$ export OS_INTERFACE=public
$ export OS_IDENTITY_API_VERSION=3
```



## 5.9 Aufgabenteil



# Aufgabenteil – Provider

Konfigurieren Sie einen OpenStack Provider.

Konfigurieren Sie einen OpenStack Provider mit den folgenden Werten:

```
auth_url: https://api-1.betacloud.de:5000/v3
user_name: tux
password: b1s
region: betacloud-1
user_domain_name: b1systems
tenant_name: training-terraform
project_domain_id: 7ace1917796541d2a5660dba84d3f664
```

## Musterlösung

Konfigurieren Sie einen OpenStack Provider mit den folgenden Werten:

```
auth_url: https://api-1.betacloud.de:5000/v3  
user_name: tux  
password: b1s  
region: betacloud-1  
user_domain_name: b1systems  
tenant_name: training-terraform  
project_domain_id: 7ace1917796541d2a5660dba84d3f664
```

### Provider:

Erzeugen Sie eine Datei (z. B. main.tf) mit folgendem Inhalt:

```
provider "openstack" {  
    auth_url      = "https://api-1.betacloud.de:5000/v3"  
    user_name     = "tux"  
    password      = "b1s"  
    region        = "betacloud-1"  
    user_domain_name = "b1systems"  
    tenant_name    = "training-terraform"  
    project_domain_id = "7ace1917796541d2a5660dba84d3f664"  
}
```

## 6 Terraform – Ressourcen erzeugen



### Terraform – Ressourcen erzeugen

## 6.1 Zielsetzung



# Zielsetzung

Dieses Kapitel

- zeigt beispielhaft das Erzeugen und Entfernen von Ressourcen
- erklärt das Skalieren von Ressourcen
- erläutert das Modifizieren bereits vorhandener Ressourcen
- erklärt, wie ein Bezug zwischen Ressourcen entsteht und genutzt wird

## 6.2 Compute Ressource erzeugen



# Compute Ressource erzeugen

## Beispiel für eine OpenStack Compute Ressource

```
resource "openstack_compute_instance_v2" "my_instance" {
    name          = "Meine Instanz"
    image_name    = "SLES15"
    flavor_name   = "4C-8GB-20GB"
    key_pair      = "john_doe_openstack"
    security_groups = ["ssh", "http", "icmp"]
    availability_zone = "south-2"

    network {
        name = "net-to-external"
    }
}
```

Als *Ressource* wird in Terraform die abstrakte Darstellung eines Produkts bezeichnet, das ein Cloud-Computing-Betreiber anbietet. Dazu können z. B. virtuelle Maschinen, aber auch DNS-Einträge, Netzwerke, Storage, Maschinelles Lernen uvw. gehören.

Im Beispiel wird als Ressource eine virtuelle Maschine (VM) in der OpenStack-Umgebung erzeugt. Dafür sind einige Parameter notwendig, wie z. B. das zu verwendende Betriebssystem.

Alle weiteren Beispiele und Aufgaben in dieser Schulung werden sich der Einfachheit halber auf virtuelle Maschinen begrenzen.

Auf der Folie ist eine Ressource definiert, die eine Virtuelle Maschine innerhalb einer OpenStack-Umgebung definiert. Auf das Stichwort `resource` folgt zunächst der Typ der gewünschten Ressource, im Beispiel `openstack_compute_instance_v2`, gefolgt von einem eindeutigen lokalen Namen. Hier wird Terraform also angewiesen, über den Provider `openstack` eine neue virtuelle Maschine mit folgenden Eckdaten zu erzeugen:

- Der Name der VM (oder Instanz) soll *Meine Instanz* lauten.
- Das Betriebssystem der VM soll ein SUSE Linux Enterprise Server 15 sein. Dabei gilt zu beachten, dass hier nur Images gewählt werden dürfen, die bereits in der Umge-

## 6 Terraform – Ressourcen erzeugen

bung hinterlegt wurden. In diesem Beispiel muss in der OpenStack-Umgebung also bereits ein Image mit dem Namen SLES15 existieren.

- Die VM soll 4 CPUs, 8 GB Arbeitsspeicher und 20 GB Festplattenspeicher erhalten. Auch hier muss der Administrator der OpenStack-Umgebung diese Werte über einen entsprechenden „Flavor“ bereits definiert haben.
- Mit dem `key_pair` wird angegeben, welcher öffentliche SSH-Schlüssel initial auf der VM hinterlegt wird. Diese Keys müssen ebenfalls bereits in der OpenStack-Umgebung für den entsprechenden Benutzer hinterlegt worden sein.
- Über die `security_groups` wird definiert, welche Ports in der Firewall von OpenStack freigegeben werden sollen. Auch diese „Gruppen“ müssen bereits in der OpenStack-Umgebung definiert worden sein. Im Beispiel lässt sich erahnen, dass die Ports 22 und 80 sowie das ICMP-Protokoll für diese VM in der Firewall von OpenStack freigegeben werden sollen. Ohne diese Angaben ließe sich keine SSH-Verbindung zur VM aufbauen. Web-Traffic und Ping-Versuche würden ebenfalls unterbunden.
- Jede VM innerhalb einer OpenStack Umgebung muss mindestens eine Netzwerk-Schnittstelle haben. Um dies zu gewährleisten wird ein neuer Code-Block geöffnet, der den Namen des Netzwerks trägt, an das die VM gebunden werden soll. Es ist problemlos möglich, mehrere Netzwerk-Code-Blöcke zu definieren, damit mehrere Netzwerk-Schnittstellen innerhalb der VM zu unterschiedlichen Netzen vorhanden sind. Ein Beispiel dafür folgt später in der Unterlage. Genau wie bei vielen der vorherigen Parameter müssen die Netzwerke bereits in OpenStack existieren, damit sie hier über ihren Namen abgerufen werden können.

Es sei noch angemerkt, dass der Hinweis bei einigen Parametern, dass die Werte vorab in OpenStack definiert sein müssen, nicht ganz korrekt ist. Je nach Zugriffsrechten des Benutzers, mit dem Terraform auf die Schnittstelle der OpenStack API zugreift, ist es z. B. möglich, neue Netzwerke, Images, Schlüsselpaare, Sicherheitsgruppen usw. auch zu erstellen. Dann müssen diese nicht bereits in OpenStack vorhanden sein, der Benutzer darf sie dann selber erstellen. Es kann aber weiterhin auch – wie oben beschrieben – auf bereits vorhandene Ressourcen zugegriffen werden.

Bei diesem Beispiel handelt es sich um ein Minimalbeispiel. Für alle möglichen Parameter siehe

[https://www.terraform.io/docs/providers/openstack/r/compute\\_instance\\_v2.html](https://www.terraform.io/docs/providers/openstack/r/compute_instance_v2.html)

### 6.3 Aufgabenteil



## Aufgabenteil – Compute Ressource erzeugen

- ① Loggen Sie sich in die vom Trainer bereitgestellte OpenStack-Testumgebung ein und laden Sie die Datei „OpenStack RC File v3“ herunter.
- ② Laden Sie die Datei in Ihre aktuelle Shell und geben Sie das Passwort für Ihren Benutzer ein.
- ③ Erzeugen Sie eine Datei main.tf.
- ④ Führen Sie terraform plan aus und lassen Sie die geplanten Änderungen visualisieren.
- ⑤ Führen Sie terraform apply aus und bestätigen Sie die geplanten Änderungen.
- ⑥ Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

1. Loggen Sie sich in die vom Trainer bereitgestellte OpenStack-Testumgebung ein und laden Sie die Datei „OpenStack RC File v3“ herunter. Klicken Sie dazu auf Ihren Benutzernamen oben rechts.
2. Laden Sie die Datei in Ihre aktuelle Shell und geben Sie das Passwort für Ihren Benutzer ein:

```
$ source *-openrc.sh
```

3. Erzeugen Sie eine Datei main.tf mit folgendem Inhalt:

```
resource "openstack_compute_instance_v2" "my_ftp_instance" {
    name          = "FTP Server"
    image_name    = "Debian 9"
    flavor_name   = "1C-1GB-10GB"
    key_pair      = "default"
    security_groups = ["ssh", "ftp"]
    availability_zone = "south-2"

    network {
        name = "net-to-external-terraform"
    }
}
```

## 6 Terraform – Ressourcen erzeugen

4. Führen Sie `terraform plan` aus und lassen Sie die geplanten Änderungen visualisieren.
5. Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen.
6. Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

## Musterlösung

1. Loggen Sie sich in die vom Trainer bereitgestellte OpenStack-Testumgebung ein und laden Sie die Datei „OpenStack RC File v3“ herunter. Klicken Sie dazu auf Ihren Benutzernamen oben rechts.
2. Laden Sie die Datei in Ihre aktuelle Shell und geben Sie das Passwort für Ihren Benutzer ein:

```
$ source *-openrc.sh
```

3. Erzeugen Sie eine Datei **main.tf** mit folgendem Inhalt:

```
resource "openstack_compute_instance_v2" "my_ftp_instance" {
    name          = "FTP Server"
    image_name    = "Debian 9"
    flavor_name   = "1C-1GB-10GB"
    key_pair      = "default"
    security_groups = ["ssh", "ftp"]
    availability_zone = "south-2"

    network {
        name = "net-to-external-terraform"
    }
}
```

4. Führen Sie **terraform plan** aus und lassen Sie die geplanten Änderungen visualisieren:

```
$ terraform plan
  Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will
  ↪ not be
persisted to local or remote state storage.
```

---

An execution plan has been generated and is shown below.  
Resource actions are indicated with the following symbols:  
+ create

Terraform will perform the following actions:

```
# openstack_compute_instance_v2.my_ftp_instance will be created
+ resource "openstack_compute_instance_v2" "my_ftp_instance" {
    + access_ip_v4      = (known after apply)
    + access_ip_v6      = (known after apply)
    + all_metadata      = (known after apply)
    + all_tags          = (known after apply)
    + availability_zone = "south-2"
    + flavor_id         = (known after apply)
    + flavor_name       = "1C-1GB-1GB"
    + force_delete      = false
```

## 6 Terraform – Ressourcen erzeugen

```
+ id                      = (known after apply)
+ image_id                = (known after apply)
+ image_name               = "Debian 9"
+ key_pair                 = "default"
+ name                     = "FTP Server"
+ power_state              = "active"
+ region                   = (known after apply)
+ security_groups          = [
    + "ftp",
    + "ssh",
]
+ stop_before_destroy      = false

+ network {
    + access_network = false
    + fixed_ip_v4   = (known after apply)
    + fixed_ip_v6   = (known after apply)
    + floating_ip   = (known after apply)
    + mac           = (known after apply)
    + name          = "net-to-external-terraform"
    + port          = (known after apply)
    + uuid          = (known after apply)
}
}
```

Plan: 1 to add, 0 to change, 0 to destroy.

---

Note: You didn't specify an "-out" parameter to save this plan,  
→ so Terraform  
can't guarantee that exactly these actions will be performed if  
"terraform apply" is subsequently run.

### 5. Führen Sie **terraform apply** aus und bestätigen Sie die geplanten Änderungen:

```
$ terraform apply
```

An execution plan has been generated and is shown below.  
Resource actions are indicated with the following symbols:  
+ create

Terraform will perform the following actions:

```
# openstack_compute_instance_v2.my_ftp_instance will be created
+ resource "openstack_compute_instance_v2" "my_ftp_instance" {
    + access_ip_v4          = (known after apply)
    + access_ip_v6          = (known after apply)
    + all_metadata          = (known after apply)
    + all_tags               = (known after apply)
    + availability_zone     = "south-2"
    + flavor_id              = (known after apply)
    + flavor_name            = "1C-1GB-1GB"
```

```

+ force_delete      = false
+ id               = (known after apply)
+ image_id         = (known after apply)
+ image_name       = "Debian 9"
+ key_pair         = "default"
+ name             = "FTP Server"
+ power_state      = "active"
+ region           = (known after apply)
+ security_groups  = [
    + "ftp",
    + "ssh",
]
+ stop_before_destroy = false

+ network {
    + access_network = false
    + fixed_ip_v4   = (known after apply)
    + fixed_ip_v6   = (known after apply)
    + floating_ip   = (known after apply)
    + mac            = (known after apply)
    + name           = "net-to-external-terraform"
    + port            = (known after apply)
    + uuid            = (known after apply)
}
}

```

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?

Terraform will perform the actions described above.

Only 'yes' will be accepted to approve.

Enter a value: yes

```

openstack_compute_instance_v2.my_ftp_instance: Creating...
openstack_compute_instance_v2.my_ftp_instance: Still creating...
    → [10s elapsed]
openstack_compute_instance_v2.my_ftp_instance: Still creating...
    → [20s elapsed]
openstack_compute_instance_v2.my_ftp_instance: Still creating...
    → [30s elapsed]
openstack_compute_instance_v2.my_ftp_instance: Creation complete
    → after 35s [id=73bfcb4c-fb40-4ea8-84e1-0441d694bddf]

```

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

## 6. Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

## 6.4 Mehrere Compute Ressourcen erzeugen



# Mehrere Compute Ressourcen erzeugen

## Beispiel für mehrere OpenStack Compute Ressourcen

```
resource "openstack_compute_instance_v2" "my_instance" {
    name          = "Meine Instanz"
    image_name    = "SLES15"
    flavor_name   = "4C-8GB-20GB"
    key_pair      = "john_doe_openstack"
    security_groups = ["ssh", "http", "icmp"]
    availability_zone = "south-2"
    count         = 2

    network {
        name = "net-to-external"
    }
}
```

Im Beispiel wurde der Ressourcen-Definition ein weiterer Parameter „count“ hinzugefügt. Dieser Parameter ist über Terraform selbst in alle möglichen Ressourcen integriert. Über ihn lässt sich sehr einfach die Anzahl der Ressourcen skalieren. In diesem Beispiel würden nun zwei virtuelle Maschinen mit den selben Einstellungen erzeugt werden.

## 6.5 Aufgabenteil



# Aufgabenteil – Mehrere Compute Ressourcen erzeugen

Es sollen drei Compute Ressourcen mit gleicher Konfiguration angelegt werden.

- ① Modifizieren Sie Ihre Datei `main.tf` und ergänzen Sie die Zeile „`count...`“.
- ② Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren.
- ③ Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen.
- ④ Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

Es sollen drei Compute Ressourcen mit gleicher Konfiguration angelegt werden.

1. Modifizieren Sie Ihre Datei `main.tf` und ergänzen Sie die Zeile „`count...`“:

```
resource "openstack_compute_instance_v2" "my_ftp_instance" {
    name          = "FTP Server"
    image_name    = "Debian 9"
    flavor_name   = "1C-1GB-10GB"
    key_pair      = "default"
    security_groups = ["ssh", "ftp"]
    availability_zone = "south-2"
    count         = 3

    network {
        name = "net-to-external-terraform"
    }
}
```

2. Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren.
3. Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen.

## 6 Terraform – Ressourcen erzeugen

4. Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

## Musterlösung

**Es sollen drei Compute Ressourcen mit gleicher Konfiguration angelegt werden.**

**1. Modifizieren Sie Ihre Datei `main.tf` und ergänzen Sie die Zeile „`count...`“:**

```
resource "openstack_compute_instance_v2" "my_ftp_instance" {
    name          = "FTP Server"
    image_name    = "Debian 9"
    flavor_name   = "1C-1GB-10GB"
    key_pair      = "default"
    security_groups = ["ssh", "ftp"]
    count         = 3

    network {
        name = "net-to-external-terraform"
    }
}
```

**2. Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren:**

```
$ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will
    ↪ not be
persisted to local or remote state storage.
```

```
openstack_compute_instance_v2.my_ftp_instance[0]: Refreshing
    ↪ state... [id=358bab14-4a5d-4d8e-9ad9-8d39d9501ef4]
```

---

An execution plan has been generated and is shown below.  
Resource actions are indicated with the following symbols:

+ create

Terraform will perform the following actions:

```
# openstack_compute_instance_v2.my_ftp_instance[1] will be
    ↪ created
+ resource "openstack_compute_instance_v2" "my_ftp_instance" {
    + access_ip_v4          = (known after apply)
    + access_ip_v6          = (known after apply)
    + all_metadata          = (known after apply)
    + all_tags               = (known after apply)
    + availability_zone     = "south-2"
    + flavor_id              = (known after apply)
    + flavor_name            = "1C-1GB-10GB"
    + force_delete           = false
    + id                     = (known after apply)
    + image_id                = (known after apply)
    + image_name              = "Debian 9"
    + key_pair                = "default"
```

## 6 Terraform – Ressourcen erzeugen

```
+ name                  = "FTP Server"
+ power_state           = "active"
+ region                = (known after apply)
+ security_groups        = [
    + "ftp",
    + "ssh",
]
+ stop_before_destroy   = false

+ network {
    + access_network = false
    + fixed_ip_v4   = (known after apply)
    + fixed_ip_v6   = (known after apply)
    + floating_ip   = (known after apply)
    + mac           = (known after apply)
    + name          = "net-to-external-training-terraform"
    + port           = (known after apply)
    + uuid           = (known after apply)
}
}

# openstack_compute_instance_v2.my_ftp_instance[2] will be
    ↪ created
+ resource "openstack_compute_instance_v2" "my_ftp_instance" {
    + access_ip_v4      = (known after apply)
    + access_ip_v6      = (known after apply)
    + all_metadata       = (known after apply)
    + all_tags          = (known after apply)
    + availability_zone = "south-2"
    + flavor_id         = (known after apply)
    + flavor_name        = "1C-1GB-10GB"
    + force_delete       = false
    + id                = (known after apply)
    + image_id          = (known after apply)
    + image_name         = "Debian 9"
    + key_pair          = "default"
    + name              = "FTP Server"
    + power_state        = "active"
    + region             = (known after apply)
    + security_groups    = [
        + "ftp",
        + "ssh",
    ]
    + stop_before_destroy = false

    + network {
        + access_network = false
        + fixed_ip_v4   = (known after apply)
        + fixed_ip_v6   = (known after apply)
        + floating_ip   = (known after apply)
        + mac           = (known after apply)
        + name          = "net-to-external-training-terraform"
        + port           = (known after apply)
        + uuid           = (known after apply)
    }
}
```

```

    }
}
```

Plan: 2 to add, 0 to change, 0 to destroy.

---

Note: You didn't specify an "-out" parameter to save this plan,  
   ↳ so Terraform  
 can't guarantee that exactly these actions will be performed if  
 "terraform apply" is subsequently run.

### 3. Führen Sie **terraform apply** aus und bestätigen Sie die geplanten Änderungen:

```
$ terraform apply
openstack_compute_instance_v2.my_ftp_instance[0]: Refreshing
  ↳ state... [id=358bab14-4a5d-4d8e-9ad9-8d39d9501ef4]
```

An execution plan has been generated and is shown below.  
 Resource actions are indicated with the following symbols:  
 + create

Terraform will perform the following actions:

```
# openstack_compute_instance_v2.my_ftp_instance[1] will be
  ↳ created
+ resource "openstack_compute_instance_v2" "my_ftp_instance" {
    + access_ip_v4      = (known after apply)
    + access_ip_v6      = (known after apply)
    + all_metadata      = (known after apply)
    + all_tags          = (known after apply)
    + availability_zone = "south-2"
    + flavor_id         = (known after apply)
    + flavor_name        = "1C-1GB-10GB"
    + force_delete       = false
    + id                = (known after apply)
    + image_id          = (known after apply)
    + image_name         = "Debian 9"
    + key_pair           = "default"
    + name               = "FTP Server"
    + power_state        = "active"
    + region             = (known after apply)
    + security_groups    = [
        + "ftp",
        + "ssh",
    ]
    + stop_before_destroy = false

+ network {
    + access_network = false
    + fixed_ip_v4    = (known after apply)
    + fixed_ip_v6    = (known after apply)
    + floating_ip     = (known after apply)
```

## 6 Terraform – Ressourcen erzeugen

```
+ mac                  = (known after apply)
+ name                 = "net-to-external-training-terraform"
+ port                 = (known after apply)
+ uuid                 = (known after apply)
}
}

# openstack_compute_instance_v2.my_ftp_instance[2] will be
  ↳ created
+ resource "openstack_compute_instance_v2" "my_ftp_instance" {
    + access_ip_v4          = (known after apply)
    + access_ip_v6          = (known after apply)
    + all_metadata          = (known after apply)
    + all_tags              = (known after apply)
    + availability_zone     = "south-2"
    + flavor_id             = (known after apply)
    + flavor_name            = "1C-1GB-10GB"
    + force_delete           = false
    + id                    = (known after apply)
    + image_id               = (known after apply)
    + image_name              = "Debian 9"
    + key_pair              = "default"
    + name                  = "FTP Server"
    + power_state            = "active"
    + region                = (known after apply)
    + security_groups        = [
        + "ftp",
        + "ssh",
    ]
    + stop_before_destroy     = false

    + network {
        + access_network = false
        + fixed_ip_v4      = (known after apply)
        + fixed_ip_v6      = (known after apply)
        + floating_ip       = (known after apply)
        + mac               = (known after apply)
        + name              = "net-to-external-training-terraform"
        + port               = (known after apply)
        + uuid               = (known after apply)
    }
}

}

Plan: 2 to add, 0 to change, 0 to destroy.
```

```
Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.
```

```
Enter a value: yes
```

```
openstack_compute_instance_v2.my_ftp_instance[1]: Creating...
openstack_compute_instance_v2.my_ftp_instance[2]: Creating...
openstack_compute_instance_v2.my_ftp_instance[1]: Still
```

```
    ↗ creating... [10s elapsed]
openstack_compute_instance_v2.my_ftp_instance[2]: Still
    ↗ creating... [10s elapsed]
openstack_compute_instance_v2.my_ftp_instance[1]: Still
    ↗ creating... [20s elapsed]
openstack_compute_instance_v2.my_ftp_instance[2]: Still
    ↗ creating... [20s elapsed]
openstack_compute_instance_v2.my_ftp_instance[2]: Creation
    ↗ complete after 25s [id=d652a974-828c-4b90-948d-80d577248278]
openstack_compute_instance_v2.my_ftp_instance[1]: Creation
    ↗ complete after 26s [id=206f0938-7230-4f8c-83fb-d4f361c55276]
```

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

4. Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

## 6.6 Modifizieren von Ressourcen



# Modifizieren von Ressourcen

## Beispiel – OpenStack Compute Ressource modifizieren

```
resource "openstack_compute_instance_v2" "my_instance" {
    name          = "Meine Instanz"
    image_name    = "SLES15"
    flavor_name   = "16C-32GB-40GB"
    key_pair      = "john_doe_openstack"
    security_groups = ["ssh", "http", "icmp"]
    availability_zone = "south-2"
    count         = 2

    network {
        name = "net-to-external"
    }
}
```

Es ist ebenfalls möglich, Werte für Ressourcen im Nachgang zu modifizieren. Je nach Provider können diese Anpassungen sogar live umgesetzt werden. In diesem Beispiel wird durch die Anpassung des Eintrags bei `flavor_name` ein anderer „Flavor“ von OpenStack gewählt, der von 8 CPUs 16 GB RAM, 20 GB Storage auf 16 CPUs, 32 GB RAM und 40 GB Storage hochskaliert. In diesem Beispiel mit OpenStack müssen die VMs dafür nicht abgebaut und wieder neu aufgebaut werden. Dies kann bei anderen Providern wiederum anders sein.

## 6.7 Aufgabenteil



# Aufgabenteil – Compute Ressourcen modifizieren 1/2

Das Image soll von Debian 9 auf Debian 10 geändert werden.

- ① Modifizieren Sie Ihre Datei `main.tf` so, dass anstelle von Debian 9 nun Debian 10 verwendet wird.
- ② Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren.
- ③ Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen.
- ④ Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

Das Image soll von Debian 9 auf Debian 10 geändert werden.

1. Modifizieren Sie Ihre Datei `main.tf` so, dass anstelle von Debian 9 nun Debian 10 verwendet wird.
2. Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren.
3. Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen.
4. Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

### Musterlösung

Das Image soll von Debian 9 auf Debian 10 geändert werden.

1. Modifizieren Sie Ihre Datei `main.tf` so, dass anstelle von Debian 9 nun Debian 10 verwendet wird:

Ändern Sie in der `main.tf` in der Zeile „`image_name...`“ den Wert von Debian 9 zu Debian 10:

```
resource "openstack_compute_instance_v2" "my_ftp_instance" {
    name          = "FTP Server"
    image_name    = "Debian 10"
    flavor_name   = "1C-1GB-10GB"
    key_pair      = "default"
    security_groups = ["ssh", "ftp"]
    count         = 3

    network {
        name = "net-to-external-terraform"
    }
}
```

2. Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren:

```
$ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will
    ↪ not be
persisted to local or remote state storage.
```

```
openstack_compute_instance_v2.my_ftp_instance[0]: Refreshing
    ↪ state... [id=358bab14-4a5d-4d8e-9ad9-8d39d9501ef4]
openstack_compute_instance_v2.my_ftp_instance[2]: Refreshing
    ↪ state... [id=d652a974-828c-4b90-948d-80d577248278]
openstack_compute_instance_v2.my_ftp_instance[1]: Refreshing
    ↪ state... [id=206f0938-7230-4f8c-83fb-d4f361c55276]
```

---

An execution plan has been generated and is shown below.  
Resource actions are indicated with the following symbols:  
-/+ destroy and then create replacement

Terraform will perform the following actions:

```
# openstack_compute_instance_v2.my_ftp_instance[0] must be
    ↪ replaced
-/+ resource "openstack_compute_instance_v2" "my_ftp_instance" {
    ~ access_ip_v4          = "10.0.0.32" -> (known after apply)
    + access_ip_v6          = (known after apply)
    ~ all_metadata           = {} -> (known after apply)
    ~ all_tags               = [] -> (known after apply)
```

```

    availability_zone      = "south-2"
~ flavor_id              = "30" -> (known after apply)
  flavor_name            = "1C-1GB-10GB"
  force_delete           = false
~ id                     =
  ↳ "358bab14-4a5d-4d8e-9ad9-8d39d9501ef4" -> (known
    ↳ after apply)
~ image_id               =
  ↳ "3511e86d-b2dd-4dfd-a41a-0452be305b8e" -> (known
    ↳ after apply)
~ image_name             = "Debian 9" -> "Debian 10" # forces
  ↳ replacement
  key_pair               = "default"
  name                   = "FTP Server"
  power_state            = "active"
~ region                 = "betacloud-1" -> (known after apply)
  security_groups        = [
    "ftp",
    "ssh",
  ]
  stop_before_destroy     = false
- tags                   = [] -> null

~ network {
  access_network          = false
~ fixed_ip_v4             = "10.0.0.32" -> (known after apply)
+ fixed_ip_v6             = (known after apply)
+ floating_ip             =
~ mac                     =
  ↳ apply
  name                   = "net-to-external-training-terraform"
+ port                    = (known after apply)
~ uuid                    =
  ↳ "71d25040-9c2b-425d-a411-f37d37d20872" -> (known
    ↳ after apply)
}
}

# openstack_compute_instance_v2.my_ftp_instance[1] must be
  ↳ replaced
-/+ resource "openstack_compute_instance_v2" "my_ftp_instance" {
  ~ access_ip_v4           = "10.0.0.8" -> (known after apply)
  + access_ip_v6             =
  ~ all_metadata            = {} -> (known after apply)
  ~ all_tags                = [] -> (known after apply)
  availability_zone         = "south-2"
  ~ flavor_id               = "30" -> (known after apply)
  flavor_name               = "1C-1GB-10GB"
  force_delete              = false
~ id                      =
  ↳ "206f0938-7230-4f8c-83fb-d4f361c55276" -> (known
    ↳ after apply)
~ image_id                =
  ↳ "3511e86d-b2dd-4dfd-a41a-0452be305b8e" -> (known

```

## 6 Terraform – Ressourcen erzeugen

```
    ↗ after apply)
~ image_name           = "Debian 9" -> "Debian 10" # forces
    ↗ replacement
key_pair               = "default"
name                  = "FTP Server"
power_state            = "active"
~ region                = "betacloud-1" -> (known after apply)
    ↗ security_groups
        "ftp",
        "ssh",
    ]
stop_before_destroy   = false
- tags                 = [] -> null

~ network {
    access_network = false
~ fixed_ip_v4         = "10.0.0.8" -> (known after apply)
+ fixed_ip_v6         = (known after apply)
+ floating_ip         = (known after apply)
~ mac                  = "fa:16:3e:4d:54:7b" -> (known after
    ↗ apply)
    name                = "net-to-external-training-terraform"
+ port                 = (known after apply)
~ uuid                 =
    ↗ "71d25040-9c2b-425d-a411-f37d37d20872" -> (known
    ↗ after apply)
}
}

# openstack_compute_instance_v2.my_ftp_instance[2] must be
    ↗ replaced
-/+ resource "openstack_compute_instance_v2" "my_ftp_instance" {
    ~ access_ip_v4          = "10.0.0.6" -> (known after apply)
+ access_ip_v6          = (known after apply)
~ all_metadata           = {} -> (known after apply)
~ all_tags               = [] -> (known after apply)
    availability_zone      = "south-2"
~ flavor_id              = "30" -> (known after apply)
    flavor_name            = "1C-1GB-10GB"
    force_delete            = false
~ id                     =
    ↗ "d652a974-828c-4b90-948d-80d577248278" -> (known
    ↗ after apply)
~ image_id               =
    ↗ "3511e86d-b2dd-4dfd-a41a-0452be305b8e" -> (known
    ↗ after apply)
~ image_name              = "Debian 9" -> "Debian 10" # forces
    ↗ replacement
key_pair               = "default"
name                  = "FTP Server"
power_state            = "active"
~ region                = "betacloud-1" -> (known after apply)
    ↗ security_groups
        "ftp",
```

```

        "ssh",
    ]
    stop_before_destroy = false
- tags                  = [] -> null

~ network {
    access_network = false
~ fixed_ip_v4     = "10.0.0.6" -> (known after apply)
+ fixed_ip_v6     = (known after apply)
+ floating_ip     = (known after apply)
~ mac              = "fa:16:3e:16:a8:8c" -> (known after
    ↪ apply)
    name            = "net-to-external-training-terraform"
+ port             = (known after apply)
~ uuid             =
    ↪ "71d25040-9c2b-425d-a411-f37d37d20872" -> (known
    ↪ after apply)
}
}

```

Plan: 3 to add, 0 to change, 3 to destroy.

---

Note: You didn't specify an "-out" parameter to save this plan,  
 ↪ so Terraform  
 can't guarantee that exactly these actions will be performed if  
 "terraform apply" is subsequently run.

### 3. Führen Sie **terraform apply** aus und bestätigen Sie die geplanten Änderungen:

```
$ terraform apply
openstack_compute_instance_v2.my_ftp_instance[0]: Refreshing
    ↪ state... [id=358bab14-4a5d-4d8e-9ad9-8d39d9501ef4]
openstack_compute_instance_v2.my_ftp_instance[2]: Refreshing
    ↪ state... [id=d652a974-828c-4b90-948d-80d577248278]
openstack_compute_instance_v2.my_ftp_instance[1]: Refreshing
    ↪ state... [id=206f0938-7230-4f8c-83fb-d4f361c55276]
```

An execution plan has been generated and is shown below.  
 Resource actions are indicated with the following symbols:  
 -/+ destroy and then create replacement

Terraform will perform the following actions:

```
# openstack_compute_instance_v2.my_ftp_instance[0] must be
    ↪ replaced
-/+ resource "openstack_compute_instance_v2" "my_ftp_instance" {
    ~ access_ip_v4          = "10.0.0.32" -> (known after apply)
    + access_ip_v6          = (known after apply)
    ~ all_metadata           = {} -> (known after apply)
    ~ all_tags               = [] -> (known after apply)
    availability_zone       = "south-2"
```

## 6 Terraform – Ressourcen erzeugen

```
    ~ flavor_id          = "30" -> (known after apply)
      flavor_name       = "1C-1GB-10GB"
      force_delete      = false
    ~ id                 =
      ↗ "358bab14-4a5d-4d8e-9ad9-8d39d9501ef4" -> (known
      ↗ after apply)
    ~ image_id           =
      ↗ "3511e86d-b2dd-4dfd-a41a-0452be305b8e" -> (known
      ↗ after apply)
    ~ image_name         = "Debian 9" -> "Debian 10" # forces
      ↗ replacement
    key_pair             = "default"
      name               = "FTP Server"
      power_state        = "active"
  ~ region              = "betacloud-1" -> (known after apply)
  security_groups      = [
    "ftp",
    "ssh",
  ]
  stop_before_destroy  = false
- tags                = [] -> null

~ network {
  access_network = false
  ~ fixed_ip_v4   = "10.0.0.32" -> (known after apply)
  + fixed_ip_v6   = (known after apply)
  + floating_ip   = (known after apply)
  ~ mac            = "fa:16:3e:b9:38:6a" -> (known after
    ↗ apply)
  name             = "net-to-external-training-terraform"
  + port           = (known after apply)
  ~ uuid           =
    ↗ "71d25040-9c2b-425d-a411-f37d37d20872" -> (known
    ↗ after apply)
}
}

# openstack_compute_instance_v2.my_ftp_instance[1] must be
  ↗ replaced
-/+ resource "openstack_compute_instance_v2" "my_ftp_instance" {
  ~ access_ip_v4        = "10.0.0.8" -> (known after apply)
  + access_ip_v6        = (known after apply)
  ~ all_metadata        = {} -> (known after apply)
  ~ all_tags            = [] -> (known after apply)
  availability_zone     = "south-2"
  ~ flavor_id           = "30" -> (known after apply)
  flavor_name           = "1C-1GB-10GB"
  force_delete          = false
  ~ id                 =
    ↗ "206f0938-7230-4f8c-83fb-d4f361c55276" -> (known
    ↗ after apply)
  ~ image_id           =
    ↗ "3511e86d-b2dd-4dfd-a41a-0452be305b8e" -> (known
    ↗ after apply)
```

```

~ image_name          = "Debian 9" -> "Debian 10" # forces
  ↏ replacement
key_pair              = "default"
name                  = "FTP Server"
power_state            = "active"
~ region               = "betacloud-1" -> (known after apply)
security_groups        = [
  "ftp",
  "ssh",
]
stop_before_destroy   = false
- tags                 = [] -> null

~ network {
  access_network = false
~ fixed_ip_v4      = "10.0.0.8" -> (known after apply)
+ fixed_ip_v6      = (known after apply)
+ floating_ip      = (known after apply)
~ mac                = "fa:16:3e:4d:54:7b" -> (known after
  ↏ apply)
  name              = "net-to-external-training-terraform"
+ port               = (known after apply)
~ uuid                =
  ↏ "71d25040-9c2b-425d-a411-f37d37d20872" -> (known
  ↏ after apply)
}
}

# openstack_compute_instance_v2.my_ftp_instance[2] must be
  ↏ replaced
-/+ resource "openstack_compute_instance_v2" "my_ftp_instance" {
  ~ access_ip_v4      = "10.0.0.6" -> (known after apply)
  + access_ip_v6      = (known after apply)
  ~ all_metadata       = {} -> (known after apply)
  ~ all_tags           = [] -> (known after apply)
  availability_zone   = "south-2"
  ~ flavor_id          = "30" -> (known after apply)
  flavor_name          = "1C-1GB-10GB"
  force_delete         = false
  ~ id                 =
    ↏ "d652a974-828c-4b90-948d-80d577248278" -> (known
    ↏ after apply)
  ~ image_id            =
    ↏ "3511e86d-b2dd-4dfd-a41a-0452be305b8e" -> (known
    ↏ after apply)
  ~ image_name          = "Debian 9" -> "Debian 10" # forces
  ↏ replacement
  key_pair              = "default"
  name                  = "FTP Server"
  power_state            = "active"
  ~ region               = "betacloud-1" -> (known after apply)
  security_groups        = [
    "ftp",
    "ssh",
  ]
}

```

## 6 Terraform – Ressourcen erzeugen

```
]
  stop_before_destroy = false
- tags                  = [] -> null

~ network {
    access_network = false
~ fixed_ip_v4      = "10.0.0.6" -> (known after apply)
+ fixed_ip_v6      = (known after apply)
+ floating_ip      = (known after apply)
~ mac                = "fa:16:3e:16:a8:8c" -> (known after
    ↪ apply)
    name              = "net-to-external-training-terraform"
+ port                = (known after apply)
~ uuid                =
    ↪ "71d25040-9c2b-425d-a411-f37d37d20872" -> (known
    ↪ after apply)
}
}
```

Plan: 3 to add, 0 to change, 3 to destroy.

Do you want to perform these actions?

Terraform will perform the actions described above.

Only 'yes' will be accepted to approve.

Enter a value: yes

```
openstack_compute_instance_v2.my_ftp_instance[2]: Destroying...
  ↪ [id=d652a974-828c-4b90-948d-80d577248278]
openstack_compute_instance_v2.my_ftp_instance[0]: Destroying...
  ↪ [id=358bab14-4a5d-4d8e-9ad9-8d39d9501ef4]
openstack_compute_instance_v2.my_ftp_instance[1]: Destroying...
  ↪ [id=206f0938-7230-4f8c-83fb-d4f361c55276]
openstack_compute_instance_v2.my_ftp_instance[2]: Still
  ↪ destroying... [id=d652a974-828c-4b90-948d-80d577248278, 10s
  ↪ elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ destroying... [id=358bab14-4a5d-4d8e-9ad9-8d39d9501ef4, 10s
  ↪ elapsed]
openstack_compute_instance_v2.my_ftp_instance[1]: Still
  ↪ destroying... [id=206f0938-7230-4f8c-83fb-d4f361c55276, 10s
  ↪ elapsed]
openstack_compute_instance_v2.my_ftp_instance[1]: Destruction
  ↪ complete after 11s
openstack_compute_instance_v2.my_ftp_instance[0]: Destruction
  ↪ complete after 11s
openstack_compute_instance_v2.my_ftp_instance[0]: Creating...
openstack_compute_instance_v2.my_ftp_instance[1]: Creating...
openstack_compute_instance_v2.my_ftp_instance[2]: Destruction
  ↪ complete after 11s
openstack_compute_instance_v2.my_ftp_instance[2]: Creating...
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [10s elapsed]
openstack_compute_instance_v2.my_ftp_instance[1]: Still
```

```
    ↗ creating... [10s elapsed]
openstack_compute_instance_v2.my_ftp_instance[2]: Still
    ↗ creating... [10s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [20s elapsed]
openstack_compute_instance_v2.my_ftp_instance[1]: Still
    ↗ creating... [20s elapsed]
openstack_compute_instance_v2.my_ftp_instance[2]: Still
    ↗ creating... [20s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [30s elapsed]
openstack_compute_instance_v2.my_ftp_instance[1]: Still
    ↗ creating... [30s elapsed]
openstack_compute_instance_v2.my_ftp_instance[2]: Still
    ↗ creating... [30s elapsed]
openstack_compute_instance_v2.my_ftp_instance[1]: Creation
    ↗ complete after 34s [id=4daf127a-9eeb-4672-9091-f7a015d1b4d9]
openstack_compute_instance_v2.my_ftp_instance[0]: Creation
    ↗ complete after 34s [id=65cdc401-4f17-418d-a6e2-1047573673fa]
openstack_compute_instance_v2.my_ftp_instance[2]: Creation
    ↗ complete after 35s [id=4678c380-9034-4168-82a1-7a9c7a10fc2e]
```

4. Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.





## Aufgabenteil – Compute Ressourcen modifizieren 2/2

Anstatt drei kleiner Instanzen soll nur eine größere Instanz (2C-2GB-10GB) verbleiben.

- 5 Modifizieren Sie Ihre Datei `main.tf` so, dass nun anstatt dreier kleiner Instanzen eine größere Instanz verbleibt.
- 6 Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren.
- 7 Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen.
- 8 Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

Anstatt drei kleiner Instanzen soll nur eine größere Instanz (2C-2GB-10GB) verbleiben.

5. Modifizieren Sie Ihre Datei `main.tf` so, dass nun anstatt dreier kleiner Instanzen eine größere Instanz verbleibt.
6. Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren.
7. Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen.
8. Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

## Musterlösung

**Anstatt drei kleiner Instanzen soll nur eine größere Instanz (2C-2GB-10GB) verbleiben.**

**5. Modifizieren Sie Ihre Datei `main.tf` so, dass nun anstatt dreier kleiner Instanzen eine größere Instanz verbleibt:**

Passen Sie erneut die `main.tf` an und ändern Sie folgende zwei Zeilen:

- „`flavor_name`...“: Wert von 1C-1GB-10GB zu 2C-2GB-10GB
- „`count`...“: Wert von 3 zu 1

```
resource "openstack_compute_instance_v2" "my_ftp_instance" {
    name          = "FTP Server"
    image_name    = "Debian 10"
    flavor_name   = "2C-2GB-10GB"
    key_pair      = "default"
    security_groups = ["ssh", "ftp"]
    availability_zone = "south-2"
    count         = 1

    network {
        name = "net-to-external-terraform"
    }
}
```

**6. Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren:**

```
$ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will
    ↪ not be
persisted to local or remote state storage.

openstack_compute_instance_v2.my_ftp_instance[2]: Refreshing
    ↪ state... [id=4678c380-9034-4168-82a1-7a9c7a10fc2e]
openstack_compute_instance_v2.my_ftp_instance[1]: Refreshing
    ↪ state... [id=4daf127a-9eeb-4672-9091-f7a015d1b4d9]
openstack_compute_instance_v2.my_ftp_instance[0]: Refreshing
    ↪ state... [id=65cdc401-4f17-418d-a6e2-1047573673fa]
```

```
-----
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
```

- ~ update in-place
- destroy

Terraform will perform the following actions:

```
# openstack_compute_instance_v2.my_ftp_instance[0] will be
    ↪ updated in-place
```

```

~ resource "openstack_compute_instance_v2" "my_ftp_instance" {
    access_ip_v4          = "10.0.0.6"
    all_metadata           = {}
    all_tags               = []
    availability_zone     = "south-2"
    flavor_id              = "30"
    ~ flavor_name           = "1C-1GB-10GB" -> "2C-2GB-10GB"
    force_delete            = false
    id                     =
        ↪ "65cdc401-4f17-418d-a6e2-1047573673fa"
    image_id               =
        ↪ "86777aef-8f22-456b-82ea-092f1e8523f9"
    image_name              = "Debian 10"
    key_pair                = "default"
    name                    = "FTP Server"
    power_state              = "active"
    region                  = "betacloud-1"
    security_groups          = [
        "ftp",
        "ssh",
    ]
    stop_before_destroy      = false
    tags                     = []

    network {
        access_network = false
        fixed_ip_v4    = "10.0.0.6"
        mac             = "fa:16:3e:db:b0:94"
        name            = "net-to-external-training-terraform"
        uuid            =
            ↪ "71d25040-9c2b-425d-a411-f37d37d20872"
    }
}

# openstack_compute_instance_v2.my_ftp_instance[1] will be
#   ↪ destroyed
- resource "openstack_compute_instance_v2" "my_ftp_instance" {
    - access_ip_v4          = "10.0.0.18" -> null
    - all_metadata           = {} -> null
    - all_tags               = [] -> null
    - availability_zone     = "south-2" -> null
    - flavor_id              = "30" -> null
    - flavor_name             = "1C-1GB-10GB" -> null
    - force_delete            = false -> null
    - id                     =
        ↪ "4daf127a-9eeb-4672-9091-f7a015d1b4d9" -> null
    - image_id               =
        ↪ "86777aef-8f22-456b-82ea-092f1e8523f9" -> null
    - image_name              = "Debian 10" -> null
    - key_pair                = "default" -> null
    - name                    = "FTP Server" -> null
    - power_state              = "active" -> null
    - region                  = "betacloud-1" -> null
    - security_groups          = [

```

## 6 Terraform – Ressourcen erzeugen

```
- "ftp",
- "ssh",
] -> null
- stop_before_destroy = false -> null
- tags = [] -> null

- network {
    - access_network = false -> null
    - fixed_ip_v4 = "10.0.0.18" -> null
    - mac = "fa:16:3e:09:e2:d5" -> null
    - name = "net-to-external-training-terraform"
        ↪ -> null
    - uuid =
        ↪ "71d25040-9c2b-425d-a411-f37d37d20872" -> null
}
}

# openstack_compute_instance_v2.my_ftp_instance[2] will be
    ↪ destroyed
- resource "openstack_compute_instance_v2" "my_ftp_instance" {
    - access_ip_v4 = "10.0.0.5" -> null
    - all_metadata = {} -> null
    - all_tags = [] -> null
    - availability_zone = "south-2" -> null
    - flavor_id = "30" -> null
    - flavor_name = "1C-1GB-10GB" -> null
    - force_delete = false -> null
    - id =
        ↪ "4678c380-9034-4168-82a1-7a9c7a10fc2e" -> null
    - image_id =
        ↪ "86777aef-8f22-456b-82ea-092f1e8523f9" -> null
    - image_name = "Debian 10" -> null
    - key_pair = "default" -> null
    - name = "FTP Server" -> null
    - power_state = "active" -> null
    - region = "betacloud-1" -> null
    - security_groups = [
        - "ftp",
        - "ssh",
    ] -> null
    - stop_before_destroy = false -> null
    - tags = [] -> null

    - network {
        - access_network = false -> null
        - fixed_ip_v4 = "10.0.0.5" -> null
        - mac = "fa:16:3e:e7:0a:89" -> null
        - name = "net-to-external-training-terraform"
            ↪ -> null
        - uuid =
            ↪ "71d25040-9c2b-425d-a411-f37d37d20872" -> null
    }
}
}
```

```
Plan: 0 to add, 1 to change, 2 to destroy.
```

Note: You didn't specify an "-out" parameter to save this plan,  
 ↪ so Terraform  
 can't guarantee that exactly these actions will be performed if  
 "terraform apply" is subsequently run.

## 7. Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen:

```
$ terraform apply
openstack_compute_instance_v2.my_ftp_instance[0]: Refreshing
    ↪ state... [id=65cdc401-4f17-418d-a6e2-1047573673fa]
openstack_compute_instance_v2.my_ftp_instance[2]: Refreshing
    ↪ state... [id=4678c380-9034-4168-82a1-7a9c7a10fc2e]
openstack_compute_instance_v2.my_ftp_instance[1]: Refreshing
    ↪ state... [id=4daf127a-9eeb-4672-9091-f7a015d1b4d9]
```

An execution plan has been generated and is shown below.  
 Resource actions are indicated with the following symbols:  
 ~ update in-place  
 - destroy

Terraform will perform the following actions:

```
# openstack_compute_instance_v2.my_ftp_instance[0] will be
    ↪ updated in-place
~ resource "openstack_compute_instance_v2" "my_ftp_instance" {
    access_ip_v4          = "10.0.0.6"
    all_metadata          = {}
    all_tags              = []
    availability_zone     = "south-2"
    flavor_id              = "30"
    ~ flavor_name          = "1C-1GB-10GB" -> "2C-2GB-10GB"
    force_delete           = false
    id                     =
        ↪ "65cdc401-4f17-418d-a6e2-1047573673fa"
    image_id               =
        ↪ "86777aef-8f22-456b-82ea-092f1e8523f9"
    image_name             = "Debian 10"
    key_pair               = "default"
    name                   = "FTP Server"
    power_state            = "active"
    region                 = "betacloud-1"
    security_groups         = [
        "ftp",
        "ssh",
    ]
    stop_before_destroy     = false
    tags                   = []

    network {
```

## 6 Terraform – Ressourcen erzeugen

```
access_network = false
fixed_ip_v4    = "10.0.0.6"
mac            = "fa:16:3e:db:b0:94"
name           = "net-to-external-training-terraform"
uuid           =
    ↪ "71d25040-9c2b-425d-a411-f37d37d20872"
}

}

# openstack_compute_instance_v2.my_ftp_instance[1] will be
    ↪ destroyed
- resource "openstack_compute_instance_v2" "my_ftp_instance" {
    - access_ip_v4      = "10.0.0.18" -> null
    - all_metadata       = {} -> null
    - all_tags          = [] -> null
    - availability_zone = "south-2" -> null
    - flavor_id         = "30" -> null
    - flavor_name        = "1C-1GB-10GB" -> null
    - force_delete       = false -> null
    - id                =
        ↪ "4daf127a-9eeb-4672-9091-f7a015d1b4d9" -> null
    - image_id          =
        ↪ "86777aef-8f22-456b-82ea-092f1e8523f9" -> null
    - image_name         = "Debian 10" -> null
    - key_pair          = "default" -> null
    - name              = "FTP Server" -> null
    - power_state        = "active" -> null
    - region             = "betacloud-1" -> null
    - security_groups   = [
        - "ftp",
        - "ssh",
    ] -> null
    - stop_before_destroy = false -> null
    - tags               = [] -> null

    - network {
        - access_network = false -> null
        - fixed_ip_v4   = "10.0.0.18" -> null
        - mac           = "fa:16:3e:09:e2:d5" -> null
        - name           = "net-to-external-training-terraform"
            ↪ -> null
        - uuid           =
            ↪ "71d25040-9c2b-425d-a411-f37d37d20872" -> null
    }
}

# openstack_compute_instance_v2.my_ftp_instance[2] will be
    ↪ destroyed
- resource "openstack_compute_instance_v2" "my_ftp_instance" {
    - access_ip_v4      = "10.0.0.5" -> null
    - all_metadata       = {} -> null
    - all_tags          = [] -> null
    - availability_zone = "south-2" -> null
    - flavor_id         = "30" -> null
```

```

- flavor_name      = "1C-1GB-10GB" -> null
- force_delete     = false -> null
- id               =
  ↗ "4678c380-9034-4168-82a1-7a9c7a10fc2e" -> null
- image_id         =
  ↗ "86777aef-8f22-456b-82ea-092f1e8523f9" -> null
- image_name       = "Debian 10" -> null
- key_pair         = "default" -> null
- name             = "FTP Server" -> null
- power_state      = "active" -> null
- region           = "betacloud-1" -> null
- security_groups  = [
  - "ftp",
  - "ssh",
]
] -> null
- stop_before_destroy = false -> null
- tags              = [] -> null

- network {
  - access_network = false -> null
  - fixed_ip_v4   = "10.0.0.5" -> null
  - mac            = "fa:16:3e:e7:0a:89" -> null
  - name           = "net-to-external-training-terraform"
    ↗ -> null
  - uuid            =
    ↗ "71d25040-9c2b-425d-a411-f37d37d20872" -> null
}
}
}

```

Plan: 0 to add, 1 to change, 2 to destroy.

Do you want to perform these actions?  
Terraform will perform the actions described above.  
Only 'yes' will be accepted to approve.

Enter a value: yes

```

openstack_compute_instance_v2.my_ftp_instance[1]: Destroying...
  ↗ [id=4daf127a-9eeb-4672-9091-f7a015d1b4d9]
openstack_compute_instance_v2.my_ftp_instance[2]: Destroying...
  ↗ [id=4678c380-9034-4168-82a1-7a9c7a10fc2e]
openstack_compute_instance_v2.my_ftp_instance[0]: Modifying...
  ↗ [id=65cdc401-4f17-418d-a6e2-1047573673fa]
openstack_compute_instance_v2.my_ftp_instance[1]: Still
  ↗ destroying... [id=4daf127a-9eeb-4672-9091-f7a015d1b4d9, 10s
  ↗ elapsed]
openstack_compute_instance_v2.my_ftp_instance[2]: Still
  ↗ destroying... [id=4678c380-9034-4168-82a1-7a9c7a10fc2e, 10s
  ↗ elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↗ modifying... [id=65cdc401-4f17-418d-a6e2-1047573673fa, 10s
  ↗ elapsed]
openstack_compute_instance_v2.my_ftp_instance[1]: Destruction
  ↗ complete after 12s

```

## 6 Terraform – Ressourcen erzeugen

```
openstack_compute_instance_v2.my_ftp_instance[2]: Destruction
  ↳ complete after 12s
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↳ modifying... [id=65cdc401-4f17-418d-a6e2-1047573673fa, 20s
  ↳ elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↳ modifying... [id=65cdc401-4f17-418d-a6e2-1047573673fa, 30s
  ↳ elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Modifications
  ↳ complete after 31s [id=65cdc401-4f17-418d-a6e2-1047573673fa]
```

8. Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

## 6.8 Netzwerk Ressource hinzufügen



# Netzwerk Ressource hinzufügen

## Beispiel für eine OpenStack Netzwerk und Subnetz Ressource

```
resource "openstack_networking_network_v2" "my_network" {
    name          = "Netzwerk Java App"
    admin_state_up = "true"
    availability_zone_hints = ["south-2"]
}

resource "openstack_networking_subnet_v2" "my_subnet" {
    name      = "Subnetz Java App"
    network_id = openstack_networking_network_v2.my_network.id
    cidr      = "192.168.42.0/24"
    ip_version = 4
}
```

Die beiden neuen Ressourcen im Beispiel werden der Compute Ressource hinzugefügt. Sie sorgen für das Anlegen eines neuen virtuellen Netzwerks mit dazugehörigem Subnetz.

Dabei ist zu beachten, wie im zweiten Code-Block (dem Subnetz-Code) auf den ersten Code-Block verlinkt wird. Der Wert der `network_id` verweist auf die Ressource vom Typ `openstack_networking_network_v2`, dort wird wiederum die eindeutige Ressource `my_network` referenziert. Über diese Art der Schreibweise (mit Punkten getrennt) kann Terraform auf Eigenschaften bereits erzeugter Ressourcen zugreifen.

Auch wenn im ersten Code-Block der Parameter „`id`“, auf den sich die Referenz bezieht, nicht definiert ist, so ist er nach der erfolgreichen Erzeugung vorhanden, sodass anschließend das Subnetz erstellt werden kann. Über diesen Weg baut Terraform ebenfalls die Abhängigkeiten und Reihenfolgen der Ressourcen auf.

Genauere Informationen zu den einzelnen Ressourcen entnehmen Sie der offiziellen Dokumentation unter:

[https://www.terraform.io/docs/providers/openstack/r/networking\\_network\\_v2.html#attributes-reference](https://www.terraform.io/docs/providers/openstack/r/networking_network_v2.html#attributes-reference)



## 6.9 Aufgabenteil



# Aufgabenteil – Netzwerk Ressource hinzufügen 1/2

Es soll eine Netzwerk Ressource inklusive eines Subnetzes definiert werden. Anschließend soll die Compute Ressource an das neu erzeugte Netz angeschlossen werden.

- 1 Führen Sie ein `terraform destroy` aus.
- 2 Editieren Sie die Datei `main.tf` so, dass eine Netzwerk Ressource (Name des Netzes „Netzwerk FTP Freigabe“, Subnetz im Bereich `192.168.42.0/24`) hinzugefügt wird.
- 3 Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren.
- 4 Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen.
- 5 Überprüfen Sie, ob alles erfolgreich umgesetzt wurde.

Es soll eine Netzwerk Ressource inklusive eines Subnetzes definiert werden. Anschließend soll die Compute Ressource an das neu erzeugte Netz angeschlossen werden.

1. Führen Sie ein `terraform destroy` aus.
2. Editieren Sie die Datei `main.tf` so, dass eine Netzwerk Ressource (Name des Netzes „Netzwerk FTP Freigabe“, Subnetz im Bereich `192.168.42.0/24`) hinzugefügt wird.
3. Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren.
4. Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen.
5. Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

### Musterlösung

**Es soll eine Netzwerk Ressource inklusive eines Subnetzes definiert werden. Anschließend soll die Compute Ressource an das neu erzeugte Netz angeschlossen werden.**

#### 1. Führen Sie ein `terraform destroy` aus:

```
$ terraform destroy
openstack_compute_instance_v2.my_ftp_instance[0]: Refreshing
  ↪ state... [id=65cdc401-4f17-418d-a6e2-1047573673fa]
```

An execution plan has been generated and is shown below.  
Resource actions are indicated with the following symbols:  
- destroy

Terraform will perform the following actions:

```
# openstack_compute_instance_v2.my_ftp_instance[0] will be
  ↪ destroyed
- resource "openstack_compute_instance_v2" "my_ftp_instance" {
    - access_ip_v4          = "10.0.0.6" -> null
    - all_metadata           = {} -> null
    - all_tags               = [] -> null
    - availability_zone     = "south-2" -> null
    - flavor_id              = "140" -> null
    - flavor_name            = "2C-2GB-10GB" -> null
    - force_delete           = false -> null
    - id                     =
      ↪ "65cdc401-4f17-418d-a6e2-1047573673fa" -> null
    - image_id Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de
      ↪ "86777aef-8f22-456b-82ea-092f1e8523f9" -> null
    - image_name             = "Debian 10" -> null
    - key_pair               = "default" -> null
    - name                   = "FTP Server" -> null
    - power_state            = "active" -> null
    - region                 = "betacloud-1" -> null
    - security_groups        = [
        - "ftp",
        - "ssh",
      ] -> null
    - stop_before_destroy    = false -> null
    - tags                   = [] -> null

    - network {
        - access_network = false -> null
        - fixed_ip_v4   = "10.0.0.6" -> null
        - mac           = "fa:16:3e:db:b0:94" -> null
        - name          = "net-to-external-training-terraform"
          ↪ -> null
        - uuid          =
          ↪ "71d25040-9c2b-425d-a411-f37d37d20872" -> null
      }
}
```

```
Plan: 0 to add, 0 to change, 1 to destroy.

Do you really want to destroy all resources?
Terraform will destroy all your managed infrastructure, as
    → shown above.
There is no undo. Only 'yes' will be accepted to confirm.

Enter a value: yes

openstack_compute_instance_v2.my_ftp_instance[0]: Destroying...
    → [id=65cdc401-4f17-418d-a6e2-1047573673fa]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    → destroying... [id=65cdc401-4f17-418d-a6e2-1047573673fa, 10s
    → elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Destruction
    → complete after 16s

Destroy complete! Resources: 1 destroyed.
```

- 2. Editieren Sie die Datei `main.tf` so, dass eine Netzwerk Ressource (Name des Netzes „Netzwerk FTP Freigabe“, Subnet im Bereich `192.168.42.0/24`) hinzugefügt wird:**

Ergänzen Sie Ihre Datei `main.tf` um folgende Zeilen:

```
resource "openstack_networking_network_v2" "ftp_network" {
    name          = "Netzwerk FTP Freigabe"
    admin_state_up = "true"
}

resource "openstack_networking_subnet_v2" "ftp_subnet" {
    name          = "Subnetz FTP Freigabe"
    network_id   = openstack_networking_network_v2.ftp_network.id
    cidr         = "192.168.42.0/24"
    ip_version   = 4
}
```

- 3. Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren:**

```
$ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will
    → not be
persisted to local or remote state storage.
```

---

```
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
+ create
```

Terraform will perform the following actions:

## 6 Terraform – Ressourcen erzeugen

```
# openstack_compute_instance_v2.my_ftp_instance[0] will be
  ↪ created
+ resource "openstack_compute_instance_v2" "my_ftp_instance" {
    + access_ip_v4          = (known after apply)
    + access_ip_v6          = (known after apply)
    + all_metadata          = (known after apply)
    + all_tags               = (known after apply)
    + availability_zone     = "south-2"
    + flavor_id              = (known after apply)
    + flavor_name            = "2C-2GB-10GB"
    + force_delete           = false
    + id                     = (known after apply)
    + image_id               = (known after apply)
    + image_name              = "Debian 10"
    + key_pair                = "default"
    + name                   = "FTP Server"
    + power_state             = "active"
    + region                 = (known after apply)
    + security_groups         = [
        + "ftp",
        + "ssh",
    ]
    + stop_before_destroy     = false

    + network {
        + access_network = false
        + fixed_ip_v4      = (known after apply)
        + fixed_ip_v6      = (known after apply)
        + floating_ip       = (known after apply)
        + mac               = (known after apply)
        + name              = "net-to-external-training-terraform"
        + port              = (known after apply)
        + uuid              = (known after apply)
    }
}

# openstack_networking_network_v2.ftp_network will be created
+ resource "openstack_networking_network_v2" "ftp_network" {
    + admin_state_up          = true
    + all_tags                 = (known after apply)
    + availability_zone_hints = (known after apply)
    + dns_domain               = (known after apply)
    + external                  = (known after apply)
    + id                       = (known after apply)
    + mtu                      = (known after apply)
    + name                     = "Netzwerk FTP Freigabe"
    + port_security_enabled    = (known after apply)
    + qos_policy_id            = (known after apply)
    + region                   = (known after apply)
    + shared                    = (known after apply)
    + tenant_id                 = (known after apply)
    + transparent_vlan          = (known after apply)
}
```

```
# openstack_networking_subnet_v2.ftp_subnet will be created
+ resource "openstack_networking_subnet_v2" "ftp_subnet" {
    + all_tags          = (known after apply)
    + cidr              = "192.168.42.0/24"
    + enable_dhcp       = true
    + gateway_ip        = (known after apply)
    + id                = (known after apply)
    + ip_version        = 4
    + ipv6_address_mode = (known after apply)
    + ipv6_ra_mode      = (known after apply)
    + name               = "Subnetz FTP Freigabe"
    + network_id         = (known after apply)
    + no_gateway         = false
    + region             = (known after apply)
    + tenant_id          = (known after apply)

    + allocation_pool {
        + end    = (known after apply)
        + start = (known after apply)
    }

    + allocation_pools {
        + end    = (known after apply)
        + start = (known after apply)
    }
}
```

Plan: 3 to add, 0 to change, 0 to destroy.

---

Note: You didn't specify an "-out" parameter to save this plan,  
 ↪ so Terraform  
 can't guarantee that exactly these actions will be performed if  
 "terraform apply" is subsequently run.

#### 4. Führen Sie **terraform apply** aus und bestätigen Sie die geplanten Änderungen:

\$ terraform apply

An execution plan has been generated and is shown below.  
 Resource actions are indicated with the following symbols:  
 + create

Terraform will perform the following actions:

```
# openstack_compute_instance_v2.my_ftp_instance[0] will be
  ↪ created
+ resource "openstack_compute_instance_v2" "my_ftp_instance" {
    + access_ip_v4          = (known after apply)
    + access_ip_v6          = (known after apply)
    + all_metadata          = (known after apply)
```

## 6 Terraform – Ressourcen erzeugen

```
+ all_tags           = (known after apply)
+ availability_zone = "south-2"
+ flavor_id          = (known after apply)
+ flavor_name         = "2C-2GB-10GB"
+ force_delete        = false
+ id                 = (known after apply)
+ image_id            = (known after apply)
+ image_name          = "Debian 10"
+ key_pair            = "default"
+ name                = "FTP Server"
+ power_state          = "active"
+ region              = (known after apply)
+ security_groups      = [
    + "ftp",
    + "ssh",
]
+ stop_before_destroy = false

+ network {
    + access_network = false
    + fixed_ip_v4   = (known after apply)
    + fixed_ip_v6   = (known after apply)
    + floating_ip    = (known after apply)
    + mac             = (known after apply)
    + name            = "net-to-external-training-terraform"
    + port             = (known after apply)
    + uuid             = (known after apply)
}
}

# openstack_networking_network_v2.ftp_network will be created
+ resource "openstack_networking_network_v2" "ftp_network" {
    + admin_state_up      = true
    + all_tags             = (known after apply)
    + availability_zone_hints = (known after apply)
    + dns_domain           = (known after apply)
    + external              = (known after apply)
    + id                   = (known after apply)
    + mtu                  = (known after apply)
    + name                 = "Netzwerk FTP Freigabe"
    + port_security_enabled = (known after apply)
    + qos_policy_id        = (known after apply)
    + region                = (known after apply)
    + shared                 = (known after apply)
    + tenant_id              = (known after apply)
    + transparent_vlan       = (known after apply)
}

# openstack_networking_subnet_v2.ftp_subnet will be created
+ resource "openstack_networking_subnet_v2" "ftp_subnet" {
    + all_tags             = (known after apply)
    + cidr                 = "192.168.42.0/24"
    + enable_dhcp           = true
    + gateway_ip            = (known after apply)
```

```

+ id                  = (known after apply)
+ ip_version         = 4
+ ipv6_address_mode = (known after apply)
+ ipv6_ra_mode       = (known after apply)
+ name               = "Subnetz FTP Freigabe"
+ network_id         = (known after apply)
+ no_gateway          = false
+ region              = (known after apply)
+ tenant_id           = (known after apply)

+ allocation_pool {
    + end    = (known after apply)
    + start = (known after apply)
}

+ allocation_pools {
    + end    = (known after apply)
    + start = (known after apply)
}
}

```

Plan: 3 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?

Terraform will perform the actions described above.

Only 'yes' will be accepted to approve.

Enter a value: yes

```

openstack_networking_network_v2.ftp_network: Creating...
openstack_compute_instance_v2.my_ftp_instance[0]: Creating...
openstack_networking_network_v2.ftp_network: Creation complete
    ↳ after 6s [id=6af2df46-11df-4056-9361-84e8c9a27825]
openstack_networking_subnet_v2.ftp_subnet: Creating...
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↳ creating... [10s elapsed]
openstack_networking_subnet_v2.ftp_subnet: Creation complete
    ↳ after 6s [id=4138036c-2378-4d6c-8a7d-60ee4e2623e0]
openstack_compute_instance_v2.my_ftp_instance[0]: Creation
    ↳ complete after 18s [id=31f2988d-af54-4fdb-a6a6-a050ecd1ee26]

```

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.

5. Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.





## Aufgabenteil – Netzwerk Ressource hinzufügen 2/2

Es soll eine Netzwerk Ressource inklusive eines Subnetzes definiert werden. Anschließend soll die Compute Ressource an das neu erzeugte Netz angeschlossen werden.

- 6 Sorgen Sie dafür, dass die Compute Ressource „my\_ftp\_instance“ an das neu erzeugte Netz angeschlossen wird.
- 7 Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren.
- 8 Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen.
- 9 Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

Es soll eine Netzwerk Ressource inklusive eines Subnetzes definiert werden. Anschließend soll die Compute Ressource an das neu erzeugte Netz angeschlossen werden.

6. Sorgen Sie dafür, dass die Compute Ressource „my\_ftp\_instance“ an das neu erzeugte Netz angeschlossen wird.
7. Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren.
8. Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen.
9. Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

## Musterlösung

**Es soll eine Netzwerk Ressource inklusive eines Subnetzes definiert werden. Anschließend soll die Compute Ressource an das neu erzeugte Netz angeschlossen werden.**

6. Sorgen Sie dafür, dass die Compute Ressource „`my_ftp_instance`“ an das neu erzeugte Netz angeschlossen wird:

Ergänzen Sie in der Datei `main.tf` die Compute Ressource „`my_ftp_instance`“ um ein weiteres Netzwerk:

```
resource "openstack_networking_network_v2" "ftp_network" {  
    name          = "Netzwerk FTP Freigabe"  
    admin_state_up = "true"  
}  
  
resource "openstack_networking_subnet_v2" "ftp_subnet" {  
    name          = "Subnetz FTP Freigabe"  
    network_id   = openstack_networking_network_v2.ftp_network.id  
    cidr         = "192.168.42.0/24"  
    ip_version   = 4  
}  
  
resource "openstack_compute_instance_v2" "my_ftp_instance" {  
    name          = "FTP Server"  
    image_name    = "Debian 10"  
    flavor_name   = "2C-2GB-10GB"  
    key_pair      = "default"  
    security_groups = ["ssh", "ftp"]  
    count         = 1  
  
    network {  
        name = "net-to-external-terraform"  
    }  
  
    network {  
        uuid = openstack_networking_network_v2.ftp_network.id  
    }  
}
```

7. Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren:

```
$ terraform plan  
Refreshing Terraform state in-memory prior to plan...  
The refreshed state will be used to calculate this plan, but will  
    ↪ not be  
persisted to local or remote state storage.  
  
openstack_networking_network_v2.ftp_network: Refreshing state...  
    ↪ [id=6af2df46-11df-4056-9361-84e8c9a27825]  
openstack_networking_subnet_v2.ftp_subnet: Refreshing state...  
    ↪ [id=4138036c-2378-4d6c-8a7d-60ee4e2623e0]
```

```
openstack_compute_instance_v2.my_ftp_instance[0]: Refreshing
  ↳ state... [id=31f2988d-af54-4fdb-a6a6-a050ecd1ee26]
```

---

An execution plan has been generated and is shown below.  
 Resource actions are indicated with the following symbols:  
 -/+ destroy and then create replacement

Terraform will perform the following actions:

```
# openstack_compute_instance_v2.my_ftp_instance[0] must be
  ↳ replaced
-/+ resource "openstack_compute_instance_v2" "my_ftp_instance" {
    ~ access_ip_v4          = "10.0.0.5" -> (known after apply)
    + access_ip_v6          = (known after apply)
    ~ all_metadata          = {} -> (known after apply)
    ~ all_tags              = [] -> (known after apply)
    availability_zone       = "south-2"
    ~ flavor_id             = "140" -> (known after apply)
    flavor_name             = "2C-2GB-10GB"
    force_delete            = false
    ~ id                    =
      ↳ "31f2988d-af54-4fdb-a6a6-a050ecd1ee26" -> (known
      ↳ after apply)
    ~ image_id              =
      ↳ "86777aef-8f22-456b-82ea-092f1e8523f9" -> (known
      ↳ after apply)
    image_name              = "Debian 10"
    key_pair                = "default"
    name                    = "FTP Server"
    power_state              = "active"
    ~ region                = "betacloud-1" -> (known after apply)
    security_groups          = [
        "ftp",
        "ssh",
    ]
    stop_before_destroy      = false
- tags                   = [] -> null

    ~ network { # forces replacement
        access_network = false
        ~ fixed_ip_v4   = "10.0.0.5" -> (known after apply)
        + fixed_ip_v6   = (known after apply)
        + floating_ip   = (known after apply)
        ~ mac           = "fa:16:3e:72:9e:61" -> (known after
          ↳ apply)
        name           = "net-to-external-training-terraform"
        + port          = (known after apply)
        ~ uuid          =
          ↳ "71d25040-9c2b-425d-a411-f37d37d20872" -> (known
          ↳ after apply)
    }
+ network { # forces replacement
```

## 6 Terraform – Ressourcen erzeugen

```
+ access_network = false
+ fixed_ip_v4    = (known after apply)
+ fixed_ip_v6    = (known after apply)
+ floating_ip    = (known after apply)
+ mac            = (known after apply)
+ name           = (known after apply)
+ port           = (known after apply)
+ uuid           =
    ↪ "6af2df46-11df-4056-9361-84e8c9a27825" # forces
    ↪ replacement
}
}

Plan: 1 to add, 0 to change, 1 to destroy.
```

```
-----
Note: You didn't specify an "-out" parameter to save this plan,
      ↪ so Terraform
can't guarantee that exactly these actions will be performed if
"terraform apply" is subsequently run.
```

### 8. Führen Sie **terraform apply** aus und bestätigen Sie die geplanten Änderungen:

```
$ terraform apply
openstack_networking_network_v2.ftp_network: Refreshing state...
    ↪ [id=6af2df46-11df-4056-9361-84e8c9a27825]
openstack_networking_subnet_v2.ftp_subnet: Refreshing state...
    ↪ [id=4138036c-2378-4d6c-8a7d-60ee4e2623e0]
openstack_compute_instance_v2.my_ftp_instance[0]: Refreshing
    ↪ state... [id=31f2988d-af54-4fdb-a6a6-a050ecd1ee26]
```

```
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
-/+ destroy and then create replacement
```

```
Terraform will perform the following actions:
```

```
# openstack_compute_instance_v2.my_ftp_instance[0] must be
    ↪ replaced
-/+ resource "openstack_compute_instance_v2" "my_ftp_instance" {
    ~ access_ip_v4          = "10.0.0.5" -> (known after apply)
    + access_ip_v6          = (known after apply)
    ~ all_metadata          = {} -> (known after apply)
    ~ all_tags              = [] -> (known after apply)
    availability_zone       = "south-2"
    ~ flavor_id             = "140" -> (known after apply)
    flavor_name             = "2C-2GB-10GB"
    force_delete            = false
    ~ id                    =
        ↪ "31f2988d-af54-4fdb-a6a6-a050ecd1ee26" -> (known
        ↪ after apply)
    ~ image_id              =
```

```

    ↗ "86777aef-8f22-456b-82ea-092f1e8523f9" -> (known
    ↗ after apply)
  image_name          = "Debian 10"
  key_pair            = "default"
  name                = "FTP Server"
  power_state         = "active"
~ region              = "betacloud-1" -> (known after apply)
  security_groups     = [
    "ftp",
    "ssh",
  ]
  stop_before_destroy = false
- tags                = [] -> null

~ network { # forces replacement
  access_network = false
~ fixed_ip_v4      = "10.0.0.5" -> (known after apply)
+ fixed_ip_v6      = (known after apply)
+ floating_ip       = (known after apply)
~ mac               = "fa:16:3e:72:9e:61" -> (known after
  ↗ apply)
  name               = "net-to-external-training-terraform"
+ port               = (known after apply)
~ uuid               =
  ↗ "71d25040-9c2b-425d-a411-f37d37d20872" -> (known
  ↗ after apply)
}
+ network { # forces replacement
  + access_network = false
  + fixed_ip_v4      = (known after apply) Erweiterung von Wohlfarth@justiz.thueringen.de
+ fixed_ip_v6      = (known after apply)
+ floating_ip       = (known after apply)
+ mac               = (known after apply)
+ name               = (known after apply)
+ port               = (known after apply)
+ uuid               =
  ↗ "6af2df46-11df-4056-9361-84e8c9a27825" # forces
  ↗ replacement
}
}

```

Plan: 1 to add, 0 to change, 1 to destroy.

Do you want to perform these actions?

Terraform will perform the actions described above.

Only 'yes' will be accepted to approve.

Enter a value: yes

```

openstack_compute_instance_v2.my_ftp_instance[0]: Destroying...
  ↗ [id=31f2988d-af54-4fdb-a6a6-a050ecd1ee26]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↗ destroying... [id=31f2988d-af54-4fdb-a6a6-a050ecd1ee26, 10s
  ↗ elapsed]

```

## 6 Terraform – Ressourcen erzeugen

```
openstack_compute_instance_v2.my_ftp_instance[0]: Destruction
  ↪ complete after 11s
openstack_compute_instance_v2.my_ftp_instance[0]: Creating...
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [10s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [20s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [30s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [40s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [50s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [1m0s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [1m10s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [1m20s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [1m30s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [1m40s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [1m50s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [2m0s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [2m10s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [2m20s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [2m30s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [2m40s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [2m50s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [3m0s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [3m10s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [3m20s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [3m30s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [3m40s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [3m50s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [4m0s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [4m10s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
```

```
    ↗ creating... [4m20s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [4m30s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [4m40s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [4m50s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [5m0s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [5m10s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [5m20s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Creation
    ↗ complete after 5m29s
    ↗ [id=497a7287-fd97-4b10-b7a3-aea2d1ef45cd]
```

Apply complete! Resources: 1 added, 0 changed, 1 destroyed.

9. Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

## 6.10 Floating IP Ressource



# Floating IPs

## Beispiel für eine OpenStack Floating IP Ressource

```
resource "openstack_networking_floatingip_v2" "my_floating_ip" {
    pool = "external"
}

resource "openstack_compute_floatingip_associate_v2" \
    "my_floating_ip_associate" {
    floating_ip = openstack_networking_floatingip_v2.my_floating_ip.address
    instance_id = openstack_compute_instance_v2.my_instance.id
}
```

Dieses Beispiel zeigt erneut die Verkettung der verschiedene Ressourcen. Das abgebildete Beispiel ist wieder eine Ergänzung der bereits vorhandenen Ressourcen.

Beispiel für eine OpenStack Floating IP Ressource (damit die Compute Ressource durch eine öffentliche IP-Adresse erreichbar ist):

```
resource "openstack_networking_floatingip_v2" "my_floating_ip" {
    pool = "external"
}

resource "openstack_compute_floatingip_associate_v2" \
    "my_floating_ip_associate" {
    floating_ip =
        openstack_networking_floatingip_v2.my_floating_ip.address
    instance_id = openstack_compute_instance_v2.my_instance.id
}
```

Die zusätzlichen Parameter wie „address“ oder „id“ sind Parameter, die eine Ressource haben kann. Sie wurden nicht explizit durch den Anwender von Terraform gesetzt, sondern werden automatisch bei Erzeugung der Ressource gefüllt und stehen danach für die weitere Verwendung zur Verfügung.

Der Vollständigkeit halber hier der Verweis auf die offizielle Dokumentation zur Ressource:

[https://www.terraform.io/docs/providers/openstack/r/networking\\_floatingip\\_v2.html#attributes-reference](https://www.terraform.io/docs/providers/openstack/r/networking_floatingip_v2.html#attributes-reference)



## 6.11 Aufgabenteil



# Aufgabenteil – Floating IP Ressource

Die Compute Ressource „my\_ftp\_instance“ soll durch eine öffentliche IP-Adresse erreichbar gemacht werden.

- ① Ergänzen Sie die Konfiguration in der `main.tf` um einen Eintrag für eine Floating IP.
- ② Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren.
- ③ Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen.
- ④ Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

Die Compute Ressource „my\_ftp\_instance“ soll durch eine öffentliche IP-Adresse erreichbar gemacht werden.

1. Ergänzen Sie die Konfiguration in der `main.tf` um einen Eintrag für eine Floating IP.
2. Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren.
3. Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen.
4. Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

### Musterlösung

Die Compute Ressource „`my_ftp_instance`“ soll durch eine öffentliche IP-Adresse erreichbar gemacht werden.

1. Ergänzen Sie die Konfiguration in der `main.tf` um einen Eintrag für eine Floating IP:

Ergänzen Sie Ihre Datei `main.tf` um folgende Zeilen:

```
resource "openstack_networking_floatingip_v2" "ftp_floating_ip" {
    pool = "public"
}

resource "openstack_compute_floatingip_associate_v2"
    ↪ "ftp_floating_ip_associate" {
    floating_ip =
        ↪ openstack_networking_floatingip_v2.ftp_floating_ip.address
    instance_id = openstack_compute_instance_v2.my_ftp_instance.0.id
}
```

2. Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren:

```
$ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will
    ↪ not be
persisted to local or remote state storage.

openstack_networking_network_v2.ftp_network: Refreshing state...
    ↪ [id=6af2df46-11df-4056-9361-84e8c9a27825]
openstack_networking_subnet_v2.ftp_subnet: Refreshing state...
    ↪ [id=4138036c-2378-4d6c-8a7d-60ee4e2623e0]
openstack_compute_instance_v2.my_ftp_instance[0]: Refreshing
    ↪ state... [id=497a7287-fd97-4b10-b7a3-aea2d1ef45cd]
```

---

An execution plan has been generated and is shown below.  
Resource actions are indicated with the following symbols:  
+ create

Terraform will perform the following actions:

```
# ↪ openstack_compute_floatingip_associate_v2.my_floating_ip_associate
    ↪ will be created
+ resource "openstack_compute_floatingip_associate_v2"
    ↪ "my_floating_ip_associate" {
        + floating_ip = (known after apply)
        + id          = (known after apply)
        + instance_id = "497a7287-fd97-4b10-b7a3-aea2d1ef45cd"
        + region       = (known after apply)
```

```

}

# openstack_networking_floatingip_v2.my_floating_ip will be
  ↪ created
+ resource "openstack_networking_floatingip_v2"
  ↪ "my_floating_ip" {
    + address      = (known after apply)
    + all_tags     = (known after apply)
    + dns_domain   = (known after apply)
    + dns_name     = (known after apply)
    + fixed_ip     = (known after apply)
    + id           = (known after apply)
    + pool          = "external"
    + port_id      = (known after apply)
    + region        = (known after apply)
    + tenant_id    = (known after apply)
  }
}

```

Plan: 2 to add, 0 to change, 0 to destroy.

---

Note: You didn't specify an "-out" parameter to save this plan,  
 ↪ so Terraform  
 can't guarantee that exactly these actions will be performed if  
 "terraform apply" is subsequently run.

### 3. Führen Sie **terraform apply** aus und bestätigen Sie die geplanten Änderungen:

```

$ terraform apply
openstack_networking_network_v2.ftp_network: Refreshing state...
  ↪ [id=6af2df46-11df-4056-9361-84e8c9a27825]
openstack_networking_subnet_v2.ftp_subnet: Refreshing state...
  ↪ [id=4138036c-2378-4d6c-8a7d-60ee4e2623e0]
openstack_compute_instance_v2.my_ftp_instance[0]: Refreshing
  ↪ state... [id=497a7287-fd97-4b10-b7a3-aea2d1ef45cd]

```

An execution plan has been generated and is shown below.  
 Resource actions are indicated with the following symbols:  
 + create

Terraform will perform the following actions:

```

#
  ↪ openstack_compute_floatingip_associate_v2.my_floating_ip_associate
  ↪ will be created
+ resource "openstack_compute_floatingip_associate_v2"
  ↪ "my_floating_ip_associate" {
    + floating_ip = (known after apply)
    + id          = (known after apply)
    + instance_id = "497a7287-fd97-4b10-b7a3-aea2d1ef45cd"
    + region       = (known after apply)
  }

```

## 6 Terraform – Ressourcen erzeugen

```
# openstack_networking_floatingip_v2.my_floating_ip will be
  ↪ created
+ resource "openstack_networking_floatingip_v2"
  ↪ "my_floating_ip" {
    + address      = (known after apply)
    + all_tags     = (known after apply)
    + dns_domain   = (known after apply)
    + dns_name     = (known after apply)
    + fixed_ip     = (known after apply)
    + id           = (known after apply)
    + pool          = "external"
    + port_id      = (known after apply)
    + region        = (known after apply)
    + tenant_id    = (known after apply)
  }
```

Plan: 2 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?

Terraform will perform the actions described above.

Only 'yes' will be accepted to approve.

Enter a value: yes

```
openstack_compute_instance_v2.my_ftp_instance[0]: Destroying...
  ↪ [id=497a7287-fd97-4b10-b7a3-aea2d1ef45cd]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ destroying... [id=497a7287-fd97-4b10-b7a3-aea2d1ef45cd, 10s
  ↪ elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Destruction
  ↪ complete after 11s
openstack_compute_instance_v2.my_ftp_instance[0]: Creating...
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [10s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [20s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [30s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [40s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [50s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [1m0s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [1m10s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [1m20s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [1m30s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
  ↪ creating... [1m40s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
```

```

    ↗ creating... [1m50s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [2m0s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [2m10s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [2m20s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [2m30s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [2m40s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [2m50s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [3m0s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [3m10s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [3m20s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [3m30s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [3m40s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [3m50s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [4m0s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [4m10s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [4m20s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [4m30s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [4m40s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [4m50s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [5m0s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [5m10s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [5m20s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Still
    ↗ creating... [5m30s elapsed]
openstack_compute_instance_v2.my_ftp_instance[0]: Creation
    ↗ complete after 5m36s
    ↗ [id=610e75bb-61e0-4ffd-b45a-97ce86887946]
openstack_compute_floatingip_associate_v2.my_floating_ip_associate:
    ↗ Creating...
openstack_compute_floatingip_associate_v2.my_floating_ip_associate:
    ↗ Creation complete after 5s
    ↗ [id=81.163.192.93/610e75bb-61e0-4ffd-b45a-97ce86887946/]

```

## 6 Terraform – Ressourcen erzeugen

```
Apply complete! Resources: 2 added, 0 changed, 1 destroyed.
```

4. Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de

## 7 Terraform – Umgang mit Variablen und Outputs



# Terraform – Umgang mit Variablen und Outputs

## 7.1 Zielsetzung



# Zielsetzung

## Dieses Kapitel

- stellt die unterschiedlichen Datentypen vor
- erläutert Deklaration und Initialisierung von Variablen
- geht auf die Reihenfolge der Variablenverarbeitung ein
- zeigt Outputs und ihre Funktion
- erörtert Datenabfragen von Umgebungen
- erklärt den Zweck von Null-Ressourcen

## 7.2 Data Types



# Data Types

- `string`
- `number`
- `bool`
- `list` (oder auch `tuple`)
- `map` (oder auch `object`)
- `null`

Es stehen verschiedene Datentypen für Variablen (*Data Types*) zur Verfügung. Die Datentypen werden automatisch von Terraform impliziert, man kann sie aber auch in einem *Variable Block* festlegen. Mehr dazu im Folgenden.

**string** Ein `string` enthält eine Zeichenkette und ist der Standard-Datentyp, sofern kein anderer Typ gesetzt ist.

„dummy“ oder „Dies ist ein Satz“ sind Beispiele für den Inhalt solcher Variablen.

**number** Variablen vom Typ `number` können sowohl ganze Zahlen als auch rationale Zahlen enthalten. `6` oder `-0.35` sind entsprechende Beispiele für den Inhalt solcher Variablen.

**bool** Unter `bool` werden logische Werte verstanden, die entweder `true` oder `false` sein können.

**list** Mit einer `list` können mehrere Werte unter einem Namen erreichbar gemacht werden (vergleichbar mit einem Array). Eine Liste mit Portfreigaben könnte beispielsweise so aussehen:

```
[ "ssh", "http", "https" ]
```

## 7 Terraform – Umgang mit Variablen und Outputs

Manche Ressourcen in Terraform erwarten eine Liste, auch wenn Sie nur einen Wert übergeben möchten. Dies würde entsprechend so aussehen:

```
[ "ssh" ]
```

**map** Der Typ `map` ist vergleichbar mit einem Dictionary. Hier werden Key-Value-Paare definiert, z. B.

```
{name = "Peter", age = 52}
```

Dabei ist zu beachten, dass die Keys (`name`, `age`) nicht doppelt vorkommen dürfen.

**null** Mittels `null` lässt sich die Abwesenheit eines Wertes definieren. Durch Setzen eines Wertes auf `null` versucht Terraform, den Default-Wert zu nutzen. Entweder lässt man die Zuweisung eines Wertes komplett aus dem Code heraus, oder man setzt den Wert auf `null`, was den selben Effekt hat. Im Hintergrund wird der Wert dann tatsächlich auf den Default gesetzt. Die Defaults können in der Provider-Dokumentation nachgelesen werden. Falls es sich um einen notwendigen Wert handelt, wird Terraform hier eine Fehlermeldung ausgeben.

### 7.3 Variablen deklarieren und initialisieren



## Variablen deklarieren und initialisieren

Unterschiedliche Wege zum Initialisieren bzw. Deklarieren von Variablen:

- interaktiv
- Umgebungsvariablen
- Datei `terraform.tfvars`
- *Variable Block*
- ...

### Variablen per CLI übergeben

```
$ terraform -var=<variablenname>=<wert>
```

### Variablen-Datei per CLI übergeben

```
$ terraform -var-file=<pfad-zur-variablendatei>
```

Terraform bietet die Möglichkeit, Variablen an verschiedenen Stellen zu deklarieren und zu initialisieren. Dabei gibt es verschiedene Prioritäten, sodass Werte von Variablen überschrieben werden könnten, wenn sie an mehr als einer Stelle deklariert und initialisiert wurden. Die Folie zeigt nur einen Ausschnitt der tatsächlichen Möglichkeiten. Weiter unten folgt eine vollständige Aufzählung.

Niedrigste Priorität hat dabei der *interaktive Modus*. Er wird durch das Deklarieren einer Variable in einer `*.tf`-Datei ohne Initialisierung ausgelöst (sofern die Variable an keiner anderen Stelle initialisiert wird). Dies erfolgt bei einem Aufruf von `terraform plan` oder `terraform apply`. Am Prompt werden dann die Werte für die Variablen abgefragt.

Weiterhin können Sie Variablen als *Umgebungsvariablen* definieren. Dabei wird vor den Variablennamen (z. B. `my_var`) das Präfix `TF_VAR_` gesetzt. Somit lautet der Name der zu exportierenden Variablen `TF_VAR_my_var`.

Über eine Datei `terraform.tfvars` (oder auch `xyz.auto.tfvars`) lassen sich *Projektbezogene Variablen* deklarieren. Ein Beispiel dafür wäre z. B. eine `terraform.tfvars` für die Produktivumgebung, eine weitere für eine Testumgebung. `terraform.tfvars`-Dateien sind nicht in HCL geschrieben, sondern im INI-Format. So lässt sich eine solche Datei auch einfacher von anderen Programmen erzeugen, z. B. durch einen automa-

## 7 Terraform – Umgang mit Variablen und Outputs

tisierten CI/CD-Aufruf (*Continuous Integration/Continuous Delivery* bzw. *Continuous Deployment*) wie durch z. B. Jenkins oder GitLab.

Der *Variable Block* ermöglicht eine Variablendefinition innerhalb von \*.tf-Dateien. Außerdem lässt sich hier eine `description` sowie ein `default`-Wert und ein Datentyp festlegen. Der Konvention entsprechend sollte die Datei `variables.tf` heißen, dies ist allerdings nicht verpflichtend.

Auch über die Kommandozeile lassen sich Variablen setzen. Dies geschieht z. B. wie folgt:

```
$ terraform -var="my_var='dummy'"
```

Hier lassen sich ebenfalls mittels

```
$ terraform -var-file="my.tfvars"
```

\*.tfvars-Dateien übergeben, welche immer die höchstmögliche Priorität haben.

Zusätzlich gibt es noch weitere Möglichkeiten zur Definition von Variablen. Dazu zählen z. B. die \*.auto.tfvars sowie Dateien im JSON-Format. Hier erfolgt die Reihenfolge der Bearbeitung alphabetisch, z. B. aaa.auto.tfvars, bbb.auto.tfvars, aaa.auto.tfvars.json, bbb.auto.tfvars.json. Das „auto“ ist hier nur ein Hinweis darauf, dass diese Dateien von einem weiteren Tool (z. B. einem Bash-Skript) generiert wurden.

Die folgende Liste enthält alle unterschiedlichen Wege zum Initialisieren bzw. Deklarieren von Variablen:

- interaktiv
- Umgebungsvariablen
- Datei `terraform.tfvars` (INI-Format)
- Datei `terraform.tfvars.json` im JSON-Format
- Dateien `*.auto.tfvars`
- Dateien `*.auto.tfvars.json`
- *Variable Block*
- direkte Deklaration auf der Kommandozeile
- direkte Übergabe einer Datei auf der Kommandozeile



## Beispiel – *Variable Block*

### Beispiel für einen *Variable Block*

```
variable "jumphost_public_network_name" {
    default      = "net-to-external-sandbox"
    type         = string
    description  = "has_to_be_set_from_commandline"
    sensitive    = true
    validation {
        condition      = can(regex("^net-to-external",
            var.jumphost_public_network_name))
        error_message = "Network value needs to start with
            \"net-to-external\"."
    }
}
```

In diesem Beispiel für einen *Variable Block* wird die Variable `jumphost_public_network_name` gesetzt:

```
variable "jumphost_public_network_name" {
    default      = "net-to-external-sandbox"
    type         = string
    description  = "has_to_be_set_from_commandline"
    sensitive    = true
    validation {
        condition      = can(regex("^net-to-external",
            ↪ var.jumphost_public_network_name))
        error_message = "Network value needs to start with
            ↪ \"net-to-external\"."
    }
}
```

Der Datentyp ist `string` (`type =`), der Standardwert der Variablen ist `net-to-external-sandbox` (`default =`). Die „`description`“ ist eine Art Kommentar, der aber auch auf STDOUT ausgegeben wird. Dies geschieht nur, wenn kein Standardwert für die Variable gesetzt und auch an keiner anderen Stelle ein Wert definiert wurde. In solch einem Fall gibt Terraform die „`description`“ aus und fragt im Anschluss einen Wert für die Variable interaktiv ab. Mittels `sensitive` können Variablen in einem `plan` oder `apply` verborgen werden, jedoch nicht im State. Dazu später mehr. Wenn gewünscht ist es ebenso möglich eine Validieren der Variablenwerte vorzunehmen.

## 7 Terraform – Umgang mit Variablen und Outputs

Hierbei wird häufig die Funktion `can()` verwendet, die einen Fehler gekapselter Abfragen (z.B. hier mit `regex()`) unterdrückt. Somit kann hier eine selbst definierte Fehlermeldung generiert werden.

Tatsächlich ist in einem *Variable Block* kein einziger Parameter notwendig.

```
variable "jumphost_public_network_name" {  
}
```

In diesem Falle würde Terraform interaktiv nach einem Inhalt für die Variable fragen. Da die `description` nicht gesetzt ist, wird ersatzweise der Variablename auf STDOUT ausgegeben. Danach folgt ein Prompt für den gewünschten Wert der Variable. Ist jedoch ein Default-Wert gesetzt, oder der Wert wird an einer anderen Stelle definiert, so erfolgt keine interaktive Abfrage an den Benutzer.

Weitere Informationen finden Sie unter:

<https://www.terraform.io/docs/configuration/variables.html>

## 7.4 Outputs



# Outputs

- Definition in `outputs.tf`
- Ausgabe auf STDOUT nach erfolgreichem Lauf
- besondere Relevanz für Module

Die „Outputs“ geben bei jedem `terraform apply` einen JSON-kodierten Rückgabewert auf STDOUT aus. Sie werden meistens in einer Datei namens `outputs.tf` definiert. Die Outputs können z. B. von anderen Programmen weiter verarbeitet werden.

So können Outputs beispielsweise eine schnelle Übersicht für Test-Szenarien bieten, in denen sich z. B. ständig die öffentliche IP-Adresse einer VM durch das Aufbauen und wieder Entfernen ändert.

## Outputs – Beispiel

### Beispiel für eine Public IP Ressource

```
output "jumphost_public_ip" {
    value = openstack_networking_floatingip_v2.jumphost_fip.0.address
}
```

Im Beispiel wird die öffentliche IP-Adresse, die zugewiesen wurde, zurückgegeben:

```
output "jumphost_public_ip" {
    value = openstack_networking_floatingip_v2.jumphost_fip.0.address
}
```

Dazu wird eine Output-Variable `jumphost_public_ip` deklariert, die den gleichen Wert wie die Variable

`openstack_networking_floatingip_v2.jumphost_fip.0.address` erhält. Mittels `openstack_networking_floatingip_v2` wird Terraform angewiesen, in der gleichnamigen Ressource nach der spezifischen Ressource namens `jumphost_fip` zu suchen. Aufgrund der `count`-Anweisung im Ressourcen-Block beinhaltet dieser Bereich mehrere Elemente. Um auf das erste Element zuzugreifen, nutzt man wie bei Arrays numerische Werte, im Beispiel die „0“ für das erste Element. Das letzte Element `address` stellt den Key dar, um auf dessen Wert zugreifen zu können. Dies könnte z. B. „123.123.123.5“ sein. Es ist ebenfalls möglich, mit einem Output andere Datentypen wie `map` oder `list` zurückzugeben.

Hier noch einige Beispiele für das Zugreifen auf Werte aus einer Liste (auch Tuple genannt):

Rückgabe von address des dritten Eintrags der Variablen

```
openstack_networking_floatingip_v2.jumphost:  
output "jumphost_public_ip" {  
    value = openstack_networking_floatingip_v2.jumphost_fip.2.address  
}
```

Rückgabe aller address-Werte der Variablen

```
openstack_netoworking_floatingip_v2.jumphost:  
output "jumphost_public_ip" {  
    value = openstack_networking_floatingip_v2.jumphost_fip.*.address  
}
```



### Hinweis Zugriff auf alle Elemente

Ein Sternchen verweist auf alle Elemente einer Liste (bzw. eines Tuple). Es ist dabei *nicht* möglich, mehrere Sternchen innerhalb eines Variablenzugriffs zu verwenden.

Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de

## 7.5 Umstellung auf Variablen und Outputs



# Umstellung auf Variablen und Outputs

## Beispiel für eine OpenStack Compute Ressource

```
resource "openstack_compute_instance_v2" "my_instance" {
    name          = "Application Server for ${var.app}"
    image_name    = var.image
    flavor_name   = var.flavor
    key_pair      = var.keypair
    security_groups = var.security_groups
    availability_zone = var.zone

    network {
        name = "net-to-external"
    }
}
```

Im Beispiel wurden die Werte für die einzelnen Parameter wie z. B. `image_name` vollständig (oder wie bei `name` teilweise) durch Variablen ersetzt. Dies bietet den Vorteil, dass der Wert der Variablen an sehr vielen Stellen verwendet werden kann. Bei Anpassung des Wertes genügt es dann, diesen nur einmal abzuändern. Der Zugriff erfolgt hier über einen eigenen Namespace `var`. Dadurch ist es möglich, die Anführungsstriche wegzulassen. Wenn Variablen-Werte Teil eines neuen Wertes werden sollen (siehe `name`), müssen diese in geschweifte Klammern mit einem vorangestellten Dollar-Symbol verwendet werden.

Es folgt ein Beispiel:

```
resource "openstack_compute_instance_v2" "my_instance" {
    name          = "Application Server for Java Dust Finder"
    image_name    = "Debian 10"
    flavor_name   = "1C-1GB-10GB"
    key_pair      = "default"
    security_groups = ["ssh"]
    availability_zone = "south-2"

    network {
        name = "net-to-external"
    }
}
```

}

wird zu:

```
variable "image" {
  default = "Debian 10"
}

resource "openstack_compute_instance_v2" "my_instance" {
  name          = "Application Server for Java Dust Finder"
  image_name    = var.image
  flavor_name   = "1C-1GB-10GB"
  key_pair      = "default"
  security_groups = ["ssh"]
  availability_zone = "south-2"

  network {
    name = "net-to-external"
  }
}
```



## 7.6 Aufgabenteil



# Aufgabenteil – Variablendefinition im *Variable Block* 1/2

Legen Sie eine neue Compute Ressource an. Dabei sollen möglichst viele Parameter als *Variable Block* definiert werden. Erstellen Sie auch einen Output mit der ID der Ressource.

- ① Erstellen Sie eine neue Datei `main.tf` in einem Verzeichnis Ihrer Wahl, in der alle Werte der OpenStack Compute Ressource über Variablen definiert werden.
- ② Ergänzen Sie einen Output in der Datei `main.tf`, der die ID der Ressource zurückliefert.
- ③ Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren.

Legen Sie eine neue Compute Ressource an. Dabei sollen möglichst viele Parameter als *Variable Block* definiert werden. Erstellen Sie auch einen Output mit der ID der Ressource.

1. Erstellen Sie eine neue Datei `main.tf` in einem Verzeichnis Ihrer Wahl, in der alle Werte der OpenStack Compute Ressource über Variablen definiert werden.
2. Ergänzen Sie einen Output in der Datei `main.tf`, der die ID der Ressource zurückliefert.
3. Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren.

## Musterlösung

Legen Sie eine neue Compute Ressource an. Dabei sollen möglichst viele Parameter als *Variable Block* definiert werden. Erstellen Sie auch einen Output mit der ID der Ressource.

1. Erstellen Sie eine neue Datei **main.tf** in einem Verzeichnis Ihrer Wahl, in der alle Werte der OpenStack Compute Ressource über Variablen definiert werden:

### main.tf:

```
variable "app" {
    default = "Java Dust Finder"
}

variable "image" {
    default = "Debian 10"
}

variable "flavor" {
    default = "1C-1GB-10GB"
}

variable "keypair" {
    default = "default"
}

variable "security_groups" {
    default = ["ssh"]
}

variable "zone" {
    default = "south-2"
}

resource "openstack_compute_instance_v2" "my_instance" {
    name          = "Application Server for ${var.app}"
    image_name    = var.image
    flavor_name   = var.flavor
    key_pair      = var.keypair
    security_groups = var.security_groups
    availability_zone = var.zone

    network {
        name = "net-to-external"
    }
}
```

**2. Ergänzen Sie einen Output in der Datei `main.tf`, der die ID der Ressource zurückliefert:**

Zusätzlicher Eintrag in `main.tf`:

```
output "my_first_output" {
    value = openstack_compute_instance_v2.my_instance.id
}
```

**3. Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren:**

```
$ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will
    ↪ not be
persisted to local or remote state storage.
```

---

An execution plan has been generated and is shown below.  
Resource actions are indicated with the following symbols:

+ create

Terraform will perform the following actions:

```
# openstack_compute_instance_v2.my_instance will be created
+ resource "openstack_compute_instance_v2" "my_instance" {
    + access_ip_v4          = (known after apply)
    + access_ip_v6          = (known after apply)
    + all_metadata          = (known after apply)
    + all_tags               = (known after apply)
    + availability_zone     = "south-2"
    + flavor_id              = (known after apply)
    + flavor_name            = "1C-1GB-10GB"
    + force_delete           = false
    + id                     = (known after apply)
    + image_id                = (known after apply)
    + image_name              = "Debian 10"
    + key_pair                = "default"
    + name                    = "Application Server for Java Dust
        ↪ Finder"
    + power_state             = "active"
    + region                  = (known after apply)
    + security_groups         = [
        + "ssh",
    ]
    + stop_before_destroy     = false

    + network {
        + access_network = false
        + fixed_ip_v4    = (known after apply)
        + fixed_ip_v6    = (known after apply)
        + floating_ip    = (known after apply)
    }
}
```

## 7 Terraform – Umgang mit Variablen und Outputs

```
+ mac          = (known after apply)
+ name         = "net-to-external"
+ port          = (known after apply)
+ uuid          = (known after apply)
}
}
```

Plan: 1 to add, 0 to change, 0 to destroy.

---

Note: You didn't specify an "-out" parameter to save this plan,  
↪ so Terraform  
can't guarantee that exactly these actions will be performed if  
"terraform apply" is subsequently run.



## Aufgabenteil – Variablendefinition im *Variable Block* 2/2

Legen Sie eine neue Compute Ressource an. Dabei sollen möglichst viele Parameter als *Variable Block* definiert werden. Erstellen Sie auch einen Output mit der ID der Ressource.

- 4 Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen.
- 5 Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

Legen Sie eine neue Compute Ressource an. Dabei sollen möglichst viele Parameter als *Variable Block* definiert werden. Erstellen Sie auch einen Output mit der ID der Ressource.

4. Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen.
5. Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

## Musterlösung

Legen Sie eine neue Compute Ressource an. Dabei sollen möglichst viele Parameter als *Variable Block* definiert werden. Erstellen Sie auch einen Output mit der ID der Ressource.

4. Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen:

```
$ terraform apply
```

```
An execution plan has been generated and is shown below.  
Resource actions are indicated with the following symbols:  
+ create
```

```
Terraform will perform the following actions:
```

```
# openstack_compute_instance_v2.my_instance will be created  
+ resource "openstack_compute_instance_v2" "my_instance" {  
    + access_ip_v4          = (known after apply)  
    + access_ip_v6          = (known after apply)  
    + all_metadata          = (known after apply)  
    + all_tags               = (known after apply)  
    + availability_zone     = "south-2"  
    + flavor_id              = (known after apply)  
    + flavor_name            = "1C-1GB-10GB"  
    + force_delete           = false  
    + id                     = (known after apply)  
    + image_id                = (known after apply)  
    + image_name              = "Debian 10"  
    + key_pair                = "default"  
    + name                   = "Application Server for Java Dust  
        ↗ Finder"  
    + power_state             = "active"  
    + region                  = (known after apply)  
    + security_groups         = [  
        + "ssh",  
    ]  
    + stop_before_destroy     = false  
  
    + network {  
        + access_network = false  
        + fixed_ip_v4   = (known after apply)  
        + fixed_ip_v6   = (known after apply)  
        + floating_ip   = (known after apply)  
        + mac           = (known after apply)  
        + name          = "net-to-external"  
        + port           = (known after apply)  
        + uuid           = (known after apply)  
    }  
}
```

```
Plan: 1 to add, 0 to change, 0 to destroy.
```

```
Do you want to perform these actions?  
Terraform will perform the actions described above.  
Only 'yes' will be accepted to approve.  
  
Enter a value: yes  
  
openstack_compute_instance_v2.my_instance: Creating...  
openstack_compute_instance_v2.my_instance: Still creating... [10s  
    ↗ elapsed]  
openstack_compute_instance_v2.my_instance: Creation complete  
    ↗ after 19s [id=77a69d10-f2a7-4a5e-80ae-c7c8626a9b9c]  
  
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Outputs:

```
my_first_output = 77a69d10-f2a7-4a5e-80ae-c7c8626a9b9c
```

5. Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

## 7.7 Abfragen von Daten



# Abfragen von Daten

## Beispiel einer Datenabfrage

```
data "openstack_images_image_v2" "my_image_id" {  
    name          = "Centos 7"  
    most_recent   = true  
  
    properties = {  
        key = "value"  
    }  
}
```

Es gibt Situationen, in denen Werte nicht von vornherein bekannt, sondern nur in der Umgebung selber vorhanden sind. Diese können abgefragt werden – sofern der Provider dies zulässt. Dafür wird `data` genutzt.

Auch hier gilt wieder: Es gibt keine vorgeschriebenen Parameter. Diese sind immer abhängig von der entsprechenden Ressource (z. B. `openstack_images_image_v2`). Die hiermit abgefragten Daten können ähnlich wie Variablen weiter im Code verwendet werden. Der Zugriff erfolgt allerdings nicht über `var`, sondern über `data`. Um aus dem obigen Beispiel die ID eines Image zu erhalten, wäre der Zugriff folgender:

```
data.openstack_images_image_v2.my_image_id.id
```

Mehr Informationen finden Sie unter:

[https://www.terraform.io/docs/providers/openstack/d/images\\_image\\_v2.html](https://www.terraform.io/docs/providers/openstack/d/images_image_v2.html)

## 7.8 Aufgabenteil



### Aufgabenteil – Abfragen von Daten

Fragen Sie die ID des Netzwerks und die ID des Image per data ab.

- ① Erstellen Sie eine neue Datei `main.tf` in einem Verzeichnis Ihrer Wahl, in der die ID des Netzwerks und des Image per data abgefragt werden.
- ② Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren.
- ③ Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen.
- ④ Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

Fragen Sie die ID des Netzwerks und die ID des Image per data ab.

1. Erstellen Sie eine neue Datei `main.tf` in einem Verzeichnis Ihrer Wahl, in der die ID des Netzwerks und des Image per data abgefragt werden.
2. Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren.
3. Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen.
4. Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

## Musterlösung

Fragen Sie die ID des Netzwerks und die ID des Image per `data` ab.

1. Erstellen Sie eine neue Datei `main.tf` in einem Verzeichnis Ihrer Wahl, in der die ID des Netzwerks und des Image per `data` abgefragt werden:

`main.tf:`

```
data "openstack_images_image_v2" "my_image_id" {
    name      = "Debian 10"
    most_recent = true
}
```

2. Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren:

```
$ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will
    ↪ not be
persisted to local or remote state storage.

data.openstack_images_image_v2.my_image_id: Refreshing state...
-----
No changes. Infrastructure is up-to-date.

This means that Terraform did not detect any differences between
    ↪ your
configuration and real physical resources that exist. As a
    ↪ result, no
actions need to be performed.
```

3. Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen:

```
$ terraform apply
data.openstack_images_image_v2.my_image_id: Refreshing state...

Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
```

4. Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

## 7.9 Block Devices



# Block Devices

## Beispielausschnitt für Block Devices

```
block_device {
    uuid                  = data.openstack_images_image_v2.my_image_name.id
    source_type           = "image"
    destination_type     = "local"
    boot_index            = 0
    delete_on_termination = true
}

block_device {
    source_type           = "blank"
    destination_type     = "volume"
    volume_size           = 2000
    boot_index            = 1
    delete_on_termination = true
}
```

Im Beispiel ist ersichtlich, wie mittels `data` abgefragte Werte verwendet werden können:

```
block_device {
    uuid                  =
        ↗ data.openstack_images_image_v2.my_image_name.id
    source_type           = "image"
    destination_type     = "local"
    boot_index            = 0
    delete_on_termination = true
}

block_device {
    source_type           = "blank"
    destination_type     = "volume"
    volume_size           = 2000
    boot_index            = 1
    delete_on_termination = true
}
```



## 7.10 Aufgabenteil



# Aufgabenteil – Block Devices

Es sollen drei Compute Instanzen mit einem zusätzlichen Netzwerk und einem weiteren Block Storage Device von 1 GB erzeugt werden.

- ① Erstellen Sie eine neue Datei `main.tf` in einem Verzeichnis Ihrer Wahl, in der drei Compute Instanzen mit einem zusätzlichen Netzwerk und einem weiteren Block Storage Device von 1 GB definiert werden.
- ② Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren.
- ③ Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen.
- ④ Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

Es sollen drei Compute Instanzen mit einem zusätzlichen Netzwerk und einem weiteren Block Storage Device von 1 GB erzeugt werden.

1. Erstellen Sie eine neue Datei `main.tf` in einem Verzeichnis Ihrer Wahl, in der drei Compute Instanzen mit einem zusätzlichen Netzwerk und einem weiteren Block Storage Device von 1 GB definiert werden.
2. Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren.
3. Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen.
4. Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

## Musterlösung

**Es sollen 3 Compute Instanzen mit einem zusätzlichen Netzwerk und einem weiteren Block Storage Device von 1 GB erzeugt werden.**

1. Erstellen Sie eine neue Datei **main.tf** in einem Verzeichnis Ihrer Wahl, in der drei Compute Instanzen mit einem zusätzlichen Netzwerk und einem weiteren Block Storage Device von 1 GB definiert werden:

**main.tf:**

```
variable "image" {
    default = "Debian 10"
}

variable "flavor" {
    default = "1C-1GB-10GB"
}

variable "keypair" {
    default = "default"
}

variable "security_groups" {
    default = ["ssh"]
}

data "openstack_images_image_v2" "my_image_id" {
    name      = "Debian 10"
    most_recent = true
}

resource "openstack_networking_network_v2" "app_net" {
    name          = "Netzwerk App"
    admin_state_up = "true"
}

resource "openstack_networking_subnet_v2" "app_subnet" {
    name      = "Subnetz App"
    network_id = openstack_networking_network_v2.app_net.id
    cidr      = "10.0.0.0/24"
    ip_version = 4
}

resource "openstack_compute_instance_v2" "my_instance" {
    name          = "Application Server for App"
    image_name    = var.image
    flavor_name   = var.flavor
    key_pair      = var.keypair
    security_groups = var.security_groups
    availability_zone = "south-2"
    count         = 3

    network {
```

```

        name = "net-to-external"
    }

    network {
        uuid = openstack_networking_network_v2.app_net.id
    }

    block_device {
        uuid              =
            ↪ data.openstack_images_image_v2.my_image_id.id
        source_type       = "image"
        destination_type = "local"
        boot_index        = 0
        delete_on_termination = true
    }

    block_device {
        source_type       = "blank"
        destination_type = "volume"
        volume_size      = 1
        boot_index        = 1
        delete_on_termination = true
    }
}

output "my_first_output" {
    value = openstack_compute_instance_v2.my_instance.id
}

```

**2. Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren:**

```

$ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will
    ↪ not be
persisted to local or remote state storage.

```

```
data.openstack_images_image_v2.my_image_id: Refreshing state...
```

---

```
An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
+ create
```

```
Terraform will perform the following actions:
```

```
# openstack_compute_instance_v2.my_instance[0] will be created
+ resource "openstack_compute_instance_v2" "my_instance" {
    + access_ip_v4      = (known after apply)
    + access_ip_v6      = (known after apply)
    + all_metadata      = (known after apply)
```

## 7 Terraform – Umgang mit Variablen und Outputs

```
+ all_tags           = (known after apply)
+ availability_zone = "south-2"
+ flavor_id          = (known after apply)
+ flavor_name         = "1C-1GB-10GB"
+ force_delete        = false
+ id                 = (known after apply)
+ image_id            = (known after apply)
+ image_name          = "Debian 10"
+ key_pair             = "default"
+ name               = "Application Server for Java Dust
    ↪ Finder"
+ power_state          = "active"
+ region              = (known after apply)
+ security_groups      = [
    + "ssh",
]
+ stop_before_destroy = false

+ block_device {
    + boot_index          = 0
    + delete_on_termination = true
    + destination_type     = "local"
    + source_type           = "image"
    + uuid                  =
        ↪ "86777aef-8f22-456b-82ea-092f1e8523f9"
}
+ block_device {
    + boot_index          = 1
    + delete_on_termination = true
    + destination_type     = "volume"
    + source_type           = "blank"
    + volume_size           = 1
}

+ network {
    + access_network = false
    + fixed_ip_v4    = (known after apply)
    + fixed_ip_v6    = (known after apply)
    + floating_ip     = (known after apply)
    + mac              = (known after apply)
    + name             = "net-to-external"
    + port              = (known after apply)
    + uuid              = (known after apply)
}
+ network {
    + access_network = false
    + fixed_ip_v4    = (known after apply)
    + fixed_ip_v6    = (known after apply)
    + floating_ip     = (known after apply)
    + mac              = (known after apply)
    + name             = (known after apply)
    + port              = (known after apply)
    + uuid              = (known after apply)
}
```

```

}

# openstack_compute_instance_v2.my_instance[1] will be created
+ resource "openstack_compute_instance_v2" "my_instance" {
    + access_ip_v4          = (known after apply)
    + access_ip_v6          = (known after apply)
    + all_metadata          = (known after apply)
    + all_tags               = (known after apply)
    + availability_zone     = "south-2"
    + flavor_id              = (known after apply)
    + flavor_name            = "1C-1GB-10GB"
    + force_delete           = false
    + id                     = (known after apply)
    + image_id                = (known after apply)
    + image_name              = "Debian 10"
    + key_pair                = "default"
    + name                   = "Application Server for Java Dust
        ↪ Finder"
    + power_state             = "active"
    + region                  = (known after apply)
    + security_groups         = [
        + "ssh",
    ]
    + stop_before_destroy     = false

    + block_device {
        + boot_index            = 0
        + delete_on_termination = true
        + destination_type      = "local"
        + source_type            = "image"
        + uuid                   =
            ↪ "86777aef-8f22-456b-82ea-092f1e8523f9"
    }
    + block_device {
        + boot_index            = 1
        + delete_on_termination = true
        + destination_type      = "volume"
        + source_type            = "blank"
        + volume_size            = 1
    }

    + network {
        + access_network          = false
        + fixed_ip_v4            = (known after apply)
        + fixed_ip_v6            = (known after apply)
        + floating_ip             = (known after apply)
        + mac                     = (known after apply)
        + name                   = "net-to-external"
        + port                    = (known after apply)
        + uuid                   = (known after apply)
    }
    + network {
        + access_network          = false
        + fixed_ip_v4            = (known after apply)
    }
}

```

## 7 Terraform – Umgang mit Variablen und Outputs

```
+ fixed_ip_v6      = (known after apply)
+ floating_ip     = (known after apply)
+ mac              = (known after apply)
+ name             = (known after apply)
+ port             = (known after apply)
+ uuid             = (known after apply)
}
}

# openstack_compute_instance_v2.my_instance[2] will be created
+ resource "openstack_compute_instance_v2" "my_instance" {
    + access_ip_v4      = (known after apply)
    + access_ip_v6      = (known after apply)
    + all_metadata      = (known after apply)
    + all_tags          = (known after apply)
    + availability_zone = "south-2"
    + flavor_id         = (known after apply)
    + flavor_name       = "1C-1GB-10GB"
    + force_delete      = false
    + id                = (known after apply)
    + image_id          = (known after apply)
    + image_name        = "Debian 10"
    + key_pair          = "default"
    + name              = "Application Server for Java Dust
                           ↪ Finder"
    + power_state       = "active"
    + region            = (known after apply)
    + security_groups   = [
        + "ssh",
    ]
    + stop_before_destroy = false

    + block_device {
        + boot_index      = 0
        + delete_on_termination = true
        + destination_type = "local"
        + source_type      = "image"
        + uuid             =
                           ↪ "86777aef-8f22-456b-82ea-092f1e8523f9"
    }
    + block_device {
        + boot_index      = 1
        + delete_on_termination = true
        + destination_type = "volume"
        + source_type      = "blank"
        + volume_size      = 1
    }

    + network {
        + access_network = false
        + fixed_ip_v4    = (known after apply)
        + fixed_ip_v6    = (known after apply)
        + floating_ip    = (known after apply)
        + mac             = (known after apply)
    }
}
```

```

+ name          = "net-to-external"
+ port          = (known after apply)
+ uuid          = (known after apply)
}
+ network {
    + access_network = false
    + fixed_ip_v4   = (known after apply)
    + fixed_ip_v6   = (known after apply)
    + floating_ip   = (known after apply)
    + mac           = (known after apply)
    + name          = (known after apply)
    + port          = (known after apply)
    + uuid          = (known after apply)
}
}

# openstack_networking_network_v2.new_network will be created
+ resource "openstack_networking_network_v2" "new_network" {
    + admin_state_up      = true
    + all_tags            = (known after apply)
    + availability_zone_hints = (known after apply)
    + dns_domain          = (known after apply)
    + external             = (known after apply)
    + id                  = (known after apply)
    + mtu                 = (known after apply)
    + name                = "neues Netzwerk"
    + port_security_enabled = (known after apply)
    + qos_policy_id        = (known after apply)
    + region               = (known after apply)
    + shared      Persönliches Exemplar von Peter Barth@justiz.thueringen.de
    + tenant_id            = (known after apply)
    + transparent_vlan     = (known after apply)
}

# openstack_networking_subnet_v2.new_subnet will be created
+ resource "openstack_networking_subnet_v2" "new_subnet" {
    + all_tags            = (known after apply)
    + cidr                = "192.168.42.0/24"
    + enable_dhcp          = true
    + gateway_ip          = (known after apply)
    + id                  = (known after apply)
    + ip_version          = 4
    + ipv6_address_mode   = (known after apply)
    + ipv6_ra_mode         = (known after apply)
    + name                = "neues Subnetz"
    + network_id          = (known after apply)
    + no_gateway           = false
    + region               = (known after apply)
    + tenant_id            = (known after apply)

    + allocation_pool {
        + end    = (known after apply)
        + start = (known after apply)
    }
}

```

```
+ allocation_pools {  
    + end      = (known after apply)  
    + start    = (known after apply)  
}  
}
```

Plan: 5 to add, 0 to change, 0 to destroy.

```
-----  
Note: You didn't specify an "-out" parameter to save this plan,  
      ↪ so Terraform  
can't guarantee that exactly these actions will be performed if  
"terraform apply" is subsequently run.
```

### 3. Führen Sie **terraform apply** aus und bestätigen Sie die geplanten Änderungen:

```
$ terraform apply  
data.openstack_images_image_v2.my_image_id: Refreshing state...
```

```
An execution plan has been generated and is shown below.  
Resource actions are indicated with the following symbols:  
+ create
```

Terraform will perform the following actions:

```
# openstack_compute_instance_v2.my_instance[0] will be created  
+ resource "openstack_compute_instance_v2" "my_instance" {  
    + access_ip_v4          = (known after apply)  
    + access_ip_v6          = (known after apply)  
    + all_metadata          = (known after apply)  
    + all_tags              = (known after apply)  
    + availability_zone     = "south-2"  
    + flavor_id             = (known after apply)  
    + flavor_name            = "1C-1GB-10GB"  
    + force_delete           = false  
    + id                    = (known after apply)  
    + image_id               = (known after apply)  
    + image_name              = "Debian 10"  
    + key_pair                = "default"  
    + name                  = "Application Server for Java Dust  
        ↪ Finder"  
    + power_state             = "active"  
    + region                 = (known after apply)  
    + security_groups         = [  
        + "ssh",  
    ]  
    + stop_before_destroy     = false  
  
    + block_device {  
        + boot_index            = 0  
        + delete_on_termination = true
```

```

+ destination_type      = "local"
+ source_type           = "image"
+ uuid                  =
    ↪ "86777aef-8f22-456b-82ea-092f1e8523f9"
}
+ block_device {
    + boot_index          = 1
    + delete_on_termination = true
    + destination_type     = "volume"
    + source_type           = "blank"
    + volume_size          = 1
}

+ network {
    + access_network      = false
    + fixed_ip_v4         = (known after apply)
    + fixed_ip_v6         = (known after apply)
    + floating_ip         = (known after apply)
    + mac                 = (known after apply)
    + name                = "net-to-external"
    + port                = (known after apply)
    + uuid                = (known after apply)
}
+ network {
    + access_network      = false
    + fixed_ip_v4         = (known after apply)
    + fixed_ip_v6         = (known after apply)
    + floating_ip         = (known after apply)
    + mac                 = (known after apply)
    + name                = (known after apply)
    + port                = (known after apply)
    + uuid                = (known after apply)
}
}

# openstack_compute_instance_v2.my_instance[1] will be created
+ resource "openstack_compute_instance_v2" "my_instance" {
    + access_ip_v4          = (known after apply)
    + access_ip_v6          = (known after apply)
    + all_metadata          = (known after apply)
    + all_tags              = (known after apply)
    + availability_zone     = "south-2"
    + flavor_id             = (known after apply)
    + flavor_name            = "1C-1GB-10GB"
    + force_delete          = false
    + id                    = (known after apply)
    + image_id              = (known after apply)
    + image_name             = "Debian 10"
    + key_pair               = "default"
    + name                  = "Application Server for Java Dust
        ↪ Finder"
    + power_state            = "active"
    + region                = (known after apply)
    + security_groups        = [

```

## 7 Terraform – Umgang mit Variablen und Outputs

```
+ "ssh",
]
+ stop_before_destroy = false

+ block_device {
    + boot_index          = 0
    + delete_on_termination = true
    + destination_type     = "local"
    + source_type          = "image"
    + uuid                 =
        ↪ "86777aef-8f22-456b-82ea-092f1e8523f9"
}
+ block_device {
    + boot_index          = 1
    + delete_on_termination = true
    + destination_type     = "volume"
    + source_type          = "blank"
    + volume_size          = 1
}

+ network {
    + access_network = false
    + fixed_ip_v4    = (known after apply)
    + fixed_ip_v6    = (known after apply)
    + floating_ip     = (known after apply)
    + mac             = (known after apply)
    + name            = "net-to-external"
    + port             = (known after apply)
    + uuid             = (known after apply)
}
+ network {
    + access_network = false
    + fixed_ip_v4    = (known after apply)
    + fixed_ip_v6    = (known after apply)
    + floating_ip     = (known after apply)
    + mac             = (known after apply)
    + name            = (known after apply)
    + port             = (known after apply)
    + uuid             = (known after apply)
}
}

# openstack_compute_instance_v2.my_instance[2] will be created
+ resource "openstack_compute_instance_v2" "my_instance" {
    + access_ip_v4          = (known after apply)
    + access_ip_v6          = (known after apply)
    + all_metadata          = (known after apply)
    + all_tags              = (known after apply)
    + availability_zone     = "south-2"
    + flavor_id              = (known after apply)
    + flavor_name            = "1C-1GB-10GB"
    + force_delete           = false
    + id                     = (known after apply)
    + image_id               = (known after apply)
```

```

+ image_name          = "Debian 10"
+ key_pair            = "default"
+ name                = "Application Server for Java Dust
    ↪ Finder"
+ power_state         = "active"
+ region              = (known after apply)
+ security_groups     = [
    + "ssh",
]
+ stop_before_destroy = false

+ block_device {
    + boot_index          = 0
    + delete_on_termination = true
    + destination_type    = "local"
    + source_type          = "image"
    + uuid                 =
        ↪ "86777aef-8f22-456b-82ea-092f1e8523f9"
}
+ block_device {
    + boot_index          = 1
    + delete_on_termination = true
    + destination_type    = "volume"
    + source_type          = "blank"
    + volume_size          = 1
}

+ network {
    + access_network      = false
    + fixed_ip_v4         = (known after apply)
    + fixed_ip_v6         = (known after apply)
    + floating_ip         = (known after apply)
    + mac                 = (known after apply)
    + name                = "net-to-external"
    + port                = (known after apply)
    + uuid                = (known after apply)
}
+ network {
    + access_network      = false
    + fixed_ip_v4         = (known after apply)
    + fixed_ip_v6         = (known after apply)
    + floating_ip         = (known after apply)
    + mac                 = (known after apply)
    + name                = (known after apply)
    + port                = (known after apply)
    + uuid                = (known after apply)
}
}

# openstack_networking_network_v2.new_network will be created
+ resource "openstack_networking_network_v2" "new_network" {
    + admin_state_up       = true
    + all_tags             = (known after apply)
    + availability_zone_hints = (known after apply)
}

```

## 7 Terraform – Umgang mit Variablen und Outputs

```
+ dns_domain          = (known after apply)
+ external            = (known after apply)
+ id                  = (known after apply)
+ mtu                = (known after apply)
+ name                = "neues Netzwerk"
+ port_security_enabled = (known after apply)
+ qos_policy_id       = (known after apply)
+ region              = (known after apply)
+ shared               = (known after apply)
+ tenant_id           = (known after apply)
+ transparent_vlan    = (known after apply)
}

# openstack_networking_subnet_v2.new_subnet will be created
+ resource "openstack_networking_subnet_v2" "new_subnet" {
    + all_tags          = (known after apply)
    + cidr              = "192.168.42.0/24"
    + enable_dhcp        = true
    + gateway_ip         = (known after apply)
    + id                = (known after apply)
    + ip_version         = 4
    + ipv6_address_mode = (known after apply)
    + ipv6_ra_mode       = (known after apply)
    + name               = "neues Subnetz"
    + network_id          = (known after apply)
    + no_gateway          = false
    + region              = (known after apply)
    + tenant_id           = (known after apply)

    + allocation_pool {
        + end      = (known after apply)
        + start   = (known after apply)
    }
}

+ allocation_pools {
    + end    = (known after apply)
    + start = (known after apply)
}
}
```

Plan: 5 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?  
Terraform will perform the actions described above.  
Only 'yes' will be accepted to approve.

Enter a value: yes

```
openstack_compute_instance_v2.my_instance[0]: Creating...
openstack_compute_instance_v2.my_instance[1]: Creating...
openstack_compute_instance_v2.my_instance[2]: Creating...
openstack_compute_instance_v2.my_instance[1]: Still creating...
    ↗ [10s elapsed]
[ ... ]
```

```
openstack_compute_instance_v2.my_instance[1]: Creation complete
  ↗ after 5m36s [id=aede9efb-61b1-4506-94b2-bb3734bff8ad]
openstack_compute_instance_v2.my_instance[2]: Creation complete
  ↗ after 5m36s [id=535a0f26-7e2b-4514-bbad-d424c2b3c0a2]
openstack_compute_instance_v2.my_instance[0]: Still creating...
  ↗ [5m40s elapsed]
openstack_compute_instance_v2.my_instance[0]: Creation complete
  ↗ after 5m47s [id=584c3db0-7328-4233-8b07-84fc9df2f6c8]

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
```

4. Überprüfen Sie im OpenStack Webfrontend (Horizon), ob alles erfolgreich umgesetzt wurde.

## 7.11 null Ressourcen



# null Ressourcen

## Beispiel für eine null Ressource

```
resource "null_resource" "my_manual_command" {
  depends_on = [openstack_compute_floatingip_associate_v2.my_floating_ip_associate]

  provisioner "remote-exec" {
    inline = ["echo 'Hello World'"]

    connection {
      type     = "ssh"
      user     = var.user
      private_key = file('/home/tux/.ssh/id_rsa')
      host     = openstack_networking_floatingip_v2.my_floating_ip_associate.address
    }
  }
}
```

Null Ressourcen bilden eine Besonderheit in Terraform. Sie sind nicht spezifisch für eine Umgebung, da sie keinerlei Verbindung benötigen. Vielmehr schaffen sie die Möglichkeit, mittels sogenannter *Provisioner* lokal und/oder remote Shell-Befehle auszuführen und Dateien zu kopieren. Außerdem können Konfigurations-Automations-Tools wie Puppet, Chef oder Salt und weitere gestartet werden.

Nach der Erstellung einer VM könnte hiermit z. B. die Konfiguration der VM durch ein Dritt-Tool, beispielsweise Salt, getriggert werden.

Beispiel für eine null Ressource:

```
resource "null_resource" "my_manual_command" {
  depends_on = [openstack_compute_floatingip_
    ↪ associate_v2.my_floating_ip_associate]

  provisioner "remote-exec" {
    inline = ["echo 'Hello World'"]

    connection {
      type     = "ssh"
      user     = var.user
      private_key = file('/home/tux/.ssh/id_rsa')
    }
  }
}
```

```
host      = openstack_networking_floatingip_
    ↪ v2.my_floating_ip_associate.address
}
}
}
```



## 7.12 Aufgabenteil



### Aufgabenteil – null Ressource

Ergänzen Sie eine Ressource zur Ausführung eines lokalen Befehls, der nach erfolgreicher Erstellung der Compute Ressource ausgeführt wird.

- ① Fügen Sie in Ihrer `main.tf` einen Eintrag für eine Ressource hinzu, die einen lokalen Befehl ausführt (z. B. „`echo "Hello World"`“), der nach erfolgreicher Erstellung der Compute Ressource ausgeführt wird.
- ② Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren.
- ③ Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen.

Ergänzen Sie eine Ressource zur Ausführung eines lokalen Befehls, der nach erfolgreicher Erstellung der Compute Ressource ausgeführt wird.

1. Fügen Sie in Ihrer `main.tf` einen Eintrag für eine Ressource hinzu, die einen lokalen Befehl ausführt (z. B. „`echo "Hello World"`“), der nach erfolgreicher Erstellung der Compute Ressource ausgeführt wird.
2. Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren.
3. Führen Sie `terraform apply` aus und bestätigen Sie die geplanten Änderungen.

## Musterlösung

**Ergänzen Sie eine Ressource zur Ausführung eines lokalen Befehls, der nach erfolgreicher Erstellung der Compute Ressource ausgeführt wird.**

1. Fügen Sie in Ihrer `main.tf` einen Eintrag für eine Ressource hinzu, die einen lokalen Befehl ausführt (z. B. „`echo "Hello World"`“), der nach erfolgreicher Erstellung der Compute Ressource ausgeführt wird:

Ergänzen Sie Ihre Datei `main.tf` um folgende Zeilen:

```
resource "null_resource" "perform_bash_command" {
  depends_on = [openstack_compute_instance_v2.my_instance]

  provisioner "local-exec" {
    command = "echo 'Hello World'"
  }
}
```

2. Führen Sie `terraform plan` aus und lassen Sie sich die geplanten Änderungen visualisieren:

```
$ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will
  ↪ not be
persisted to local or remote state storage.

data.openstack_images_image_v2.my_image_id: Refreshing state...
openstack_networking_network_v2.new_network: Refreshing state...
  ↪ [id=f0eef688-ac88-4d3c-9b0a-730f70406e36]
openstack_networking_subnet_v2.new_subnet: Refreshing state...
  ↪ [id=24d1077a-af55-48e1-9f71-88d26f262966]
openstack_compute_instance_v2.my_instance[0]: Refreshing state...
  ↪ [id=584c3db0-7328-4233-8b07-84fc9df2f6c8]
openstack_compute_instance_v2.my_instance[1]: Refreshing state...
  ↪ [id=aede9efb-61b1-4506-94b2-bb3734bff8ad]
openstack_compute_instance_v2.my_instance[2]: Refreshing state...
  ↪ [id=535a0f26-7e2b-4514-bbad-d424c2b3c0a2]

-----
Personal Exemplar von peterwehlfarth@justizthueringen.de
```

An execution plan has been generated and is shown below.  
Resource actions are indicated with the following symbols:  
+ create

Terraform will perform the following actions:

```
# null_resource.perform_bash_command will be created
+ resource "null_resource" "perform_bash_command" {
  + id = (known after apply)
}
```

Plan: 1 to add, 0 to change, 0 to destroy.

---

Note: You didn't specify an "-out" parameter to save this plan,  
   → so Terraform  
 can't guarantee that exactly these actions will be performed if  
 "terraform apply" is subsequently run.

### 3. Führen Sie **terraform apply** aus und bestätigen Sie die geplanten Änderungen:

```
$ terraform apply
data.openstack_images_image_v2.my_image_id: Refreshing state...
openstack_networking_network_v2.new_network: Refreshing state...
  → [id=f0eef688-ac88-4d3c-9b0a-730f70406e36]
openstack_networking_subnet_v2.new_subnet: Refreshing state...
  → [id=24d1077a-af55-48e1-9f71-88d26f262966]
openstack_compute_instance_v2.my_instance[0]: Refreshing state...
  → [id=584c3db0-7328-4233-8b07-84fc9df2f6c8]
openstack_compute_instance_v2.my_instance[1]: Refreshing state...
  → [id=aede9efb-61b1-4506-94b2-bb3734bff8ad]
openstack_compute_instance_v2.my_instance[2]: Refreshing state...
  → [id=535a0f26-7e2b-4514-bbad-d424c2b3c0a2]
```

An execution plan has been generated and is shown below.  
 Resource actions are indicated with the following symbols:  
 + create

Terraform will perform the following actions:

```
# null_resource.perform_bash_command will be created
+ resource "null_resource" "perform_bash_command" {
    + id = (known after apply)
}
```

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?  
 Terraform will perform the actions described above.  
 Only 'yes' will be accepted to approve.

Enter a value: yes

```
null_resource.perform_bash_command: Creating...
null_resource.perform_bash_command: Provisioning with
  → 'local-exec'...
null_resource.perform_bash_command (local-exec): Executing:
  → ["/bin/sh" "-c" "echo 'Hello World'"]
null_resource.perform_bash_command (local-exec): Hello World
null_resource.perform_bash_command: Creation complete after 0s
  → [id=15698731555384723]
```

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

## 8 Terraform – Speichern von Zuständen



# Terraform – Speichern von Zuständen

## 8.1 Zielsetzung



# Zielsetzung

Dieses Kapitel

- liefert einen Überblick über sogenannte *States*
- erläutert den Nutzen von States

## 8.2 `terraform.tfstate`



# Die Datei `terraform.tfstate`

- != temporäre Datei
- bildet den tatsächlichen Stand ab
- gehört zum Backend
- wird bei Aktion gesperrt
- Importieren von Ressourcen
- JSON
- `terraform.tfstate.backup`

Die Datei `terraform.tfstate`, auch einfach nur *State* genannt, hilft z. B. dabei, Metadaten beizubehalten und erhöht bei größeren Deployments die Performance. Sie wird von Terraform automatisch generiert und sollte nicht manuell bearbeitet werden.

Da diese Datei den tatsächlichen Stand abbildet, sollte sie nicht gelöscht werden. Es gibt keine Möglichkeit, den gesamten Stand aus einer Umgebung auszulesen. Wurde ein Deployment erfolgreich ausgeführt und danach die `terraform.tfstate` gelöscht, so würde beim erneuten Deployment alles erneut aufgebaut werden. Ist die `terraform.tfstate` noch vorhanden, wird das Deployment nicht erneut ausgeführt. Es werden lediglich die Status der einzelnen Ressourcen aktualisiert und nichts geändert. Da diese Datei wichtig in der Interaktion mit bereits vorhandenen Deployment ist, bildet Sie einen Teil des *Backends*.

Ein *Backend* ist eine Unterstützung für ein Arbeitsverzeichnis; weiteres dazu im Kapitel „Arbeiten im Team“.

Ist ein Terraform-Lauf aktiv, wird die `terraform.tfstate` gesperrt, um doppelte Ausführungen zu vermeiden.

Weiterhin können Ressourcen in den *State* importiert werden, sofern die entsprechende Ressource dies unterstützt. Diese werden dann als Code in die `terraform.tfstate`-

Datei geschrieben. Dies kann z. B. sinnvoll sein, wenn man eine komplexe Konfiguration in z. B. einer GUI der entsprechenden Umgebung durchgeführt hat und diese in Terraform-Code überführen möchte.

Die `terraform.tfstate.backup` ist eine Kopie der `terraform.tfstate`, die zu Beginn eines neuen Terraform-Laufs erzeugt wird. Sollte in diesem Lauf etwas fehlschlagen, so ist immer noch dieses Backup des letzten erfolgreichen Laufs vorhanden.

Das Format der `terraform.tfstate` und auch der `terraform.tfstate.backup` ist JSON. So kann die Datei auch von anderen Programmen eingelesen werden. Es ließe sich beispielsweise anhand der Datei ein Inventar für Ansible oder Puppet generieren.

## 9 Terraform – Module



# Terraform – Module

## 9.1 Zielsetzung



# Zielsetzung

Dieses Kapitel

- zeigt den Aufbau von Modulen
- nennt Voraussetzungen für die Verwendung der Module
- erläutert den Nutzen von Modulen

## 9.2 Aufbau von Modulen



# Aufbau von Modulen

- nichts anderes als ein Projekt
- eigener Ordner
- `variables.tf`, `outputs.tf`

Ein *Modul* ist nichts weiter als ein weiteres Projektverzeichnis in Terraform, nur dass hier die Dateien `variables.tf` (wenn Variablen an das Modul übergeben werden sollen) und `outputs.tf` (wenn Werte aus dem Modul zurückgegeben werden sollen) notwendig sind. Module werden von anderen Projekten eingebunden, um die Funktion des Moduls zu nutzen.

Beispiel-Idee für ein Modul: Ein Consulting-Unternehmen unterstützt andere Firmen dabei, ihre Anwendungen zu einem Hyperscaler zu migrieren. Die Anwendungen selbst sind grundverschieden, allerdings nutzen alle eine Jumphost-Infrastruktur. Diese könnte in ein Modul ausgelagert werden.

Es ist außerdem möglich, ein Modul als eigenständiges Projekt zu verwenden. Im Anschluss an das vorherige Beispiel würde dann der Jumphost erstellt werden.

### 9.3 Module – Best Practices



## Module – Best Practices

- \*.`tfvars` vermeiden
- überall eindeutige Namen verwenden
- nicht im Arbeitsverzeichnis hinterlegen
- Submodule verwenden

\*.`tfvars`-Dateien sind primär nur für das Arbeitsverzeichnis gültig. Ein Beispiel:  
In `variables.tfvars` wird eine Variable `dummy` mit dem Wert `top` gesetzt. In `my_module/variables.tfvars` wird ebenfalls die Variable `dummy` gesetzt, allerdings mit dem Wert `bottom`.

```
main.tf
my_module
my_module/main.tf
my_module/variables.tfvars (dummy="bottom")
variables.tfvars (dummy="top")
```

Für alle Ressourcen ist nur der Wert `top` gültig, `bottom` wird ignoriert. Jedoch würde `bottom` gültig sein, wenn unter `variables.tfvars` die Variable `dummy` fehlen würde.

Da hier häufiger logische Fehler auftreten können, sollte man bei Modulen von der Verwendung von `variables.tfvars` abraten. Außerdem sollte ein Modul und alles innerhalb eines Moduls mit eindeutigen, ggf. sogar einzigartigen Namen versehen werden, um mögliche Probleme direkt zu vermeiden. Weiterhin ist es ebenfalls ratsam, ein Modul nicht direkt im Verzeichnis zu platzieren, in dem mit Terraform gearbeitet wird. Weiteres zu diesem Thema finden Sie im Kapitel „Arbeiten im Team“.

## 9 Terraform – Module

Wird ein Modul größer, ist es unter Umständen auch sinnvoll, dieses in weitere kleinere Submodule aufzuteilen. Je nach Projektgröße kann dies zukünftige Arbeitsaufwände reduzieren.

## 9.4 Aufgabenteil



# Aufgabenteil – Modul

- ① Erstellen Sie ein neues Projektverzeichnis.
- ② Binden Sie die Aufgabe zum Jumphost aus dem letzten Kapitel als Modul ein.

1. Erstellen Sie ein neues Projektverzeichnis.
2. Binden Sie die Aufgabe zum Jumphost aus dem letzten Kapitel als Modul ein.

## Musterlösung

**1. Erstellen Sie ein neues Projektverzeichnis:**

```
$ mkdir /home/<Benutzer>/terraform_use_modules
```

**2. Binden Sie die Aufgabe zum Jumphost aus dem letzten Kapitel als Modul ein:**

Legen Sie zunächst eine \*.tf-Datei an:

```
# vim main.tf
```

In der Datei wird der Modul-Block definiert:

```
module "jumphost" {
  source = "../terraform_project"
}
```

Anschließend muss das Verzeichnis noch initialisiert werden:

```
$ terraform init
Initializing modules...
- jumphost in ../terraform_project
```

```
Initializing the backend...
```

```
Terraform has been successfully initialized!
```

```
You may now begin working with Terraform. Try running "terraform
  ↪ plan" to see
any changes that are required for your infrastructure. All
  ↪ Terraform commands
should now work.
```

```
If you ever set or change modules or backend configuration for
  ↪ Terraform,
rerun this command to reinitialize your working directory. If you
  ↪ forget, other
commands will detect it and remind you to do so if necessary.
```

## 10 Terraform – Arbeiten im Team



# Terraform – Arbeiten im Team

## 10.1 Zielsetzung



# Zielsetzung

Dieses Kapitel liefert einen Überblick über

- die Vorteile einer Code-Versionierung
- mögliche Tool-Chains
- die Fallstricke beim Arbeiten im Team

## 10.2 Git, SVN & Co



# Git, SVN & Co

- Versionsverwaltung
- parallele Entwicklung
- Automation mittels CI/CD

Durch den Einsatz von Versionsverwaltungstools wie Git oder SVN ist es möglich, Infrastruktur zu versionieren. Da die Beschreibung in Code-Form mittels Terraform erfolgt, kann hier eine parallele Entwicklung mit mehreren Personen gleichzeitig erfolgen. Auch das automatische Ausführen von Änderungen ist möglich, indem in einer CI/CD-Pipeline (*Continuous Integration/Continuous Delivery* bzw. *Continuous Deployment*) Terraform mit dem neusten Code ausgeführt wird.

### 10.3 Wrapper Scripts



## Wrapper Scripts

- Vermeidung von händischer Interaktion
- automatisches Anlegen von Workspaces
- Sicherstellen des korrekten *State*

Keine manuelle Bestätigung der geplanten Änderungen

```
$ terraform apply -auto-approve
```

Da Terraform eher ein Mittel zum Zweck ist, wird seltener mit der Kommandozeile per Hand gearbeitet als über Scripte. Die manuelle Ausführung sollte wenn möglich auch vermieden werden, wenn viele Variablen übergeben oder mit automatisch generierten *Variable Files* gearbeitet wird. Dies übernehmen andere Tools (Ansible, GitLab CI-CD, etc.). Auch ein einfaches Bash-Script kann hier schon ausreichend sein. Ein solches kann sich z. B. auch darum kümmern, Workspaces anzulegen. Außerdem kann ein Wrapper Script zusätzliche Sicherheit in Bezug auf States geben (z. B. durch Sichern der `terraform.tfstate.backup`).

Zum Unterdrücken der manuellen Bestätigung der geplanten Änderungen bei Terraform gibt es den Schalter `-auto-approve`. Ein vollständiger Befehl sähe dann so aus:

```
$ terraform apply -auto-approve
```

## 10.4 Admin Workstation



# Admin Workstation

- einheitlicher State
- Credentials können hinterlegt werden
- größere Datenmengen müssen nicht lokal gespeichert werden
- (z. B. Terraform und Ansible für ein Deployment)

Obwohl das Konzept einer Admin Workstation nicht mehr „State of the Art“ ist, so bietet es im Falle von Terraform doch einige entscheidende Vorteile: Zum einen kann man dort den State zentral speichern. Somit entstehen keine Differenzen in der Landschaft, die durch unterschiedliche States unvermeidbar wären. Zum anderen können auf einer Admin Workstation Credentials und auch größere Datenmengen hinterlegt werden. Terraform selbst wird sehr selten alleine zum Einsatz kommen. Sollten also nach dem Bereitstellen einer Infrastruktur durch Terraform weitere Konfigurationen/Applikationen ausgerollt werden (z. B. durch Ansible), bietet es sich an, diese ebenfalls auf der Admin Workstation zu hinterlegen.

Letztendlich sind Terraform Cloud und die weiteren Bezahl-Varianten nichts anderes als eine zentrale Instanz, auf der die States und Credentials hinterlegt werden. Allerdings bieten diese Varianten noch zusätzliche Funktionen wie Rechteverwaltung, Code Compliance und weiteres.

## 11 Terraform – Alternative Technologien



# Terraform – Alternative Technologien

## 11.1 Zielsetzung



# Zielsetzung

## Dieses Kapitel

- liefert einen Überblick über alternative Lösungen sowie
- Tools mit Funktionsüberschneidungen

## 11.2 Ansible, Chef, Puppet, usw.



### Ansible, Chef, Puppet, usw.

- eher Konfiguration einer Ressource
- weniger Ressourcen-Erzeugung
- gute Ergänzung zu Terraform

Ansible, Chef oder Puppet legen den Fokus mehr auf Systemkonfiguration bei bereits vorhandenen Ressourcen. Dennoch lassen sich z. B. mittels Ansible sehr einfach NetApp-Ressourcen oder VMware-Ressourcen erzeugen. Häufig werden Tools wie Terraform in Kombination mit einem der zuvor genannten Tools eingesetzt.

### 11.3 Libraries



## Libraries

- boto (Python)
- Fog (Ruby)

Bibliotheken bilden eine alternative Möglichkeit, um Ressourcen direkt aus einer Software heraus zu erzeugen. Dies kann durchaus sinnvoll sein – Sie sollten vorher überprüfen, welche Methode für die Anforderungen besser geeignet ist.

Ein Beispiel:

Gegeben sei ein Dienstleister, der sich auf Web-Entwicklung spezialisiert hat. Dabei stellt er Portale für seine Kunden zur Verfügung, in denen diese selbstständig neue Websites aufsetzen, diese skalieren können, etc. Wenn der Dienstleister sowieso schon seine Kernarbeit auf Programmierung ausgelegt hat, ist es naheliegend, direkt eine Bibliothek zu verwenden. Ein eigenständiges Tool in eine nahezu homogene Umgebung einzuführen generiert hier mehr Arbeitsaufwand. Wiederum ist es für Unternehmen, die sich weniger mit Entwicklung beschäftigen, ggf. einfacher, ein neues Tool zu verwenden.

## 11.4 Cloudformation, Deployment Manager, Heat



# Cloudformation, Deployment Manager, Heat

- Anbieter-spezifisch
- unflexibel
- teilweise mehr Funktionen im eigenen Umfeld

Anbieter-spezifische Tools wie z. B. Cloudformation von AWS, Deployment Manager von GCP oder Heat von OpenStack bieten Anwendern schnelle und reibungslose Interaktion mit der eigenen Umgebung. Terraform ermöglicht eine Unabhängigkeit von diesen Tools bei beinahe fast identischem Funktionsumfang.

## 11.5 Pulumi



### Pulumi

- fast identisch von der Funktion
- weniger weit verbreitet
- keine DSL notwendig (Python, Go, JavaScript, TypeScript, C#)

*Pulumi* ist ein Konkurrenz-Tool zu Terraform; der Funktionsumfang ist beinahe identisch. Die größten Unterschiede liegen in der Code-Sprache sowie der Verwaltung des States. Bei Pulumi kann der Code in Python, Go, JavaScript, TypeScript oder C# geschrieben werden, auch gemischte Formen sind möglich. Außerdem stellt Pulumi für die zentrale Speicherung der States einen Dienst kostenlos zur Verfügung. Das bringt den Vorteil, dass man einfacher im Team arbeiten kann, allerdings liegt die Datenhoheit hier auch bei Pulumi.

## 12 Einführung in Ansible



# Einführung in Ansible

## 12.1 Was ist Ansible?



# Ansible – Überblick

- Werkzeug für IT-Automatisierung
- Ablaufbeschreibungen für komplexe Abläufe
- Ad-Hoc Kommandos
- Python & YAML
- Linux, Unix-artige, Windows
- Open Source Software
- 2012 veröffentlicht

Bei *Ansible* handelt es sich um ein Werkzeug, das sowohl zur Automatisierung von Prozessen auf einzelnen Systemen als auch in gesamten Systemlandschaften zum Einsatz kommen kann. Durch die Möglichkeiten der Ablaufbeschreibung von komplexen Abläufen ist es idempotent einsetzbar. Dies bedeutet, dass Ansible den beschriebenen Zustand herstellt und bei weiteren Iterationen dafür sorgt, dass genau dieser und kein anderer Zustand herrscht.

Die *Ad-Hoc-Kommandos* ermöglichen es Ihnen, einfache Aufgaben schnell und direkt ohne vorherige Konfiguration oder das Erstellen sogenannter *Playbooks* auszuführen.

Ansible ist in *Python* programmiert und muss nur auf dem Computer installiert sein, von dem aus die Systeme beschrieben werden. Auf den Zielrechnern muss lediglich Python vorhanden sein, um die mit Ansible gegebenen Anweisungen auszuführen. Eingesetzt werden kann Ansible dabei plattformübergreifend in Linux- wie auch Windowsumgebungen.

Die Konfiguration von Ansible erfolgt über *YAML*-Dateien.

Ansible wurde 2012 erstmals veröffentlicht. Als Open-Source-Lizenz wird die *GNU Public License (GPL)* Version 3 genutzt.

## Ansible Inc.

- 2013 von u. a. Michael DeHaan gegründet
- 2015 von Red Hat gekauft
- unterstützt Entwicklung
- bietet Support
- kommerzielles Produkt: *Ansible Automation Platform (AAP)* mit dem *Automation Controller*

Ansible wurde ursprünglich von Michael DeHaan entwickelt, der vorher bereits an der Entwicklung von *Cobbler* beteiligt war, einem *Provisioning Server* für die automatisierte Installation von Systemen über das Netzwerk.

2013 gründete DeHaan mit einigen Mitstreitern *Ansible Inc.*, die die Entwicklung von Ansible unterstützen und kommerziellen Support bieten sollte. Im Oktober 2015 übernahm Red Hat die Firma.

Als kommerzielles Produkt bietet Red Hat inzwischen die *Ansible Automation Platform (AAP)* an, deren *Automation Controller* vorher unter dem Namen *Ansible Tower* angeboten wurde. Dabei stützt sich die *Ansible Automation Platform* auf einige Open-Source-Projekte, u. a. AWX, auf dem der vorher angebotene *Ansible Tower* basierte. Mehr zu AWX finden Sie im Kapitel 273 ab S. 609.

## 12.2 Einsatzgebiete



# Einsatzgebiete

- Ad-Hoc Kommandos
- Systemkonfiguration
- Softwaredeployment
- Continuous Delivery
- Orchestrierung komplexer Aufgaben

Mit seinen Werkzeugen können Sie Ansible für eine Vielzahl verschiedener Aufgaben nutzen.

Für einfache Aufgaben, die keine Konfiguration erfordern, bieten sich die *Ad-Hoc Kommandos* an, die Sie direkt auf der Kommandozeile ausführen. Mit den komplexeren Werkzeugen wie *Playbooks* oder *Rollen* können ganze Systemlandschaften beschrieben und gepflegt werden. Unterstützen kann Ansible Sie auch beim Softwaredeployment sowie dem Überprüfen und Aktualisieren von Versionsständen und Konfigurationen. Bei komplexen Aufgaben der Orchestrierung von Maschinen und der Umsetzung und Automatisierung bietet Ansible eine Vielzahl an Möglichkeiten.

## Systemkonfiguration

- einzelne Änderungen über ganze Landschaft ausrollen
- Basis-Konfiguration sicherstellen
- Infrastructure as Code
- On-Demand Testumgebungen

Mit Ansible können leicht einzelne Änderungen für die gesamte Landschaft übernommen werden, da Ansible diese mit vordefinierten Bedingungen an die jeweiligen Systeme anpassen kann. Zudem können mit Ansible ganze Systementwürfe erstellt werden, die dann bei Bedarf einfach umgesetzt werden. Somit kann bestehende Infrastruktur schnell an den nötigen Bedarf angepasst werden, z. B. das Zur-Verfügung-Stellen einer bestimmten Testumgebung.



# Infrastructure as Code (IaC)

- ermöglicht die Definition und Konfiguration von physikalischen und virtuellen Umgebungen
- Beschreibung in maschinenlesbaren Definitionsdateien
- beschreibt automatisierte Abläufe zur Verwaltung und Bereitstellung von Rechenzentren
- betont nicht-physische Hardwarekonfigurationen und automatisierte Werkzeuge
- erleichtert die Verwaltung von Dateien in einer Versionskontrolle
- nicht zu verwechseln mit *Infrastructure as a Service* (IaaS)

*Infrastructure as Code* (IaC) ermöglicht die Definition und Konfiguration sowohl von physikalischen als auch virtuellen Umgebungen in maschinenlesbaren Dateien. Diese beschreiben automatisierte Abläufe zur Verwaltung und Bereitstellung von Rechenzentren. IaC betont dabei eine nicht-physische Hardwarekonfiguration und setzt auf automatisierte Werkzeuge wie Ansible. Die Verwendung von Textdateien, wie z. B. im YAML- oder INI-Format, erleichtert die Verwaltung der Daten in einer Versionskontrolle wie z. B. Git.

*Infrastructure as a Service* (IaaS) beschreibt hingegen einen Service für die Bereitstellung von virtuellen Maschinen inklusive Netzwerk und Storage in einer virtuellen Umgebung.

Andere Werkzeuge, die sich zur Definition von *Infrastructure as Code* eignen, sind z. B. *Salt*, *Terraform* und *Vagrant*. Die Grenzen zu Konfigurations-Management- und Deployment-Werkzeugen sind dabei fließend.

## Softwaredeployment – neue Software

- Tarballs verteilen und entpacken
- Pakete installieren
- Konfiguration ausrollen
- Software starten
- Monitoring aktivieren

Ansible ist in der Lage, den kompletten Software-Deployment-Vorgang idempotent durchzuführen. Das bedeutet, dass derselbe Bereitstellungsvorgang wiederholt ausgeführt werden kann, ohne zusätzliche oder unerwartete Effekte zu erzeugen.

Eine Möglichkeit ist es, eine Software als Tarball bereitzustellen, auf unterschiedliche Systemen zu kopieren und dort zu entpacken, angepasst an die jeweilige Umgebung.

Alternativ dazu ist auch die Installation als angepasstes Paket über den jeweils unterstützten Paketmanager der verschiedenen Linux-Distributionen möglich.

Danach werden benötigte Konfigurationen ausgerollt, die zum Zielsystem passen. Zum Abschluss kann die Software gestartet und ein Monitoring zur Überwachung aktiviert werden.



## Softwaredeployment – Updates

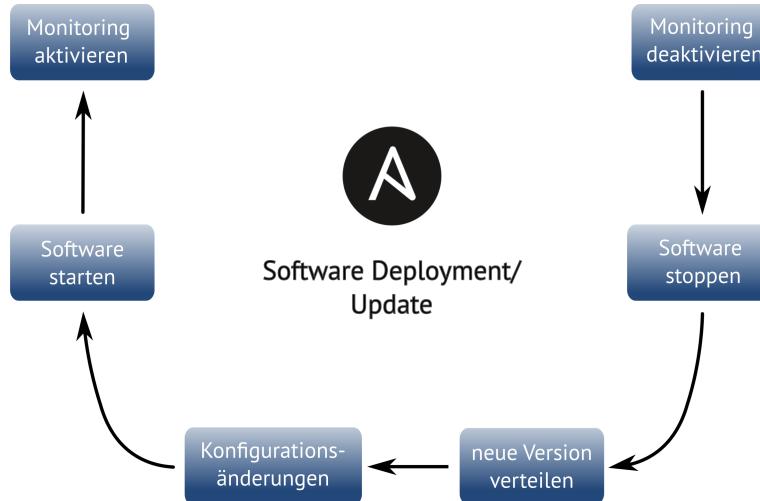


Abbildung: Softwareupdates in der Produktion

Ansible erlaubt es Ihnen, die für Softwareupdates notwendigen Prozesse zu automatisieren. Ein typisches Update könnte dabei folgende Schritte beinhalten:

1. Monitoring abschalten, um Störungen während des Aktualisierungsvorgangs zu vermeiden.
2. Software stoppen – dies ist vor dem Aktualisieren von Software immer ratsam.
3. Neue Version verteilen.
4. Konfigurationsänderungen durchführen, d. h. Konfigurationsdateien auf Zielmaschinen aktualisieren.
5. Software starten, nachdem die Software selbst und die Konfigurationsdateien aktualisiert sind.
6. Monitoring aktivieren, um die Anwendung wieder zu überwachen.

Dies ist lediglich ein allgemeines Beispiel. Konkrete Schritte können von Fall zu Fall stark variieren. Ansible bietet die Flexibilität, diese Schritte anzupassen und individuell auf Ihre Anforderungen zuzuschneiden.

# Continuous Delivery (CD)

- Softwareentwicklung
- neuer Code wird eingecheckt
- CI-Tool (z. B. Jenkins) baut und testet
- bei Erfolg rollt Ansible die neue Version automatisch aus
- Server für Server

Mit *Continuous Delivery* (CD) kann Software kontinuierlich entwickelt, getestet und automatisch bereitgestellt werden. Auch hierbei kann Ansible eine wichtige Hilfestellung leisten.

Sobald neuer Code eingecheckt wird, beispielsweise über eine Versionsverwaltung wie Git, kann ein CI-Tool (*Continuous Integration*) wie Jenkins die Software automatisiert bauen und testen.

Nach erfolgreichen Tests steuert Ansible dann den Prozess der automatisierten Bereitstellung, der die Software auf den Zielsystemen aktualisiert und – falls notwendig – Konfigurationsänderungen durchführt. Dabei ist auch ein schrittweises Ausrollen Server für Server möglich.

Ansible kann zu jedem der Schritte eine aussagekräftige Rückmeldung geben und es können Regeln definiert werden, was im Falle des Scheiterns eines Schritts oder Tests passieren soll. Durch dieses Verfahren können Sie einen reibungslosen und risikoarmen Aktualisierungsprozess gewährleisten.



# Orchestrierung komplexer Aufgaben

Beispiel: Entlastung eines Webservice durch Starten weiterer Instanzen:

- gestartet durch Monitoring Hook
- Ansible startet eine neue VM oder Container
- Deployment der Basis-Konfiguration
- Software-Deployment
- Aufnahme in den Loadbalancer

Mit Ansible lassen sich auch Aufgaben einer Orchestrierung erledigen, wie sie beispielsweise beim Aufsetzen eines Webservers mit Loadbalancer gewünscht wären:

Durch einen Monitoring Hook, d. h. die Integration mit Monitoring-Systemen, kann ein automatisierter Prozess ausgelöst werden, um auf Ereignisse wie erhöhte Last zu reagieren. Ansible würde in dem Fall eine neue VM oder einen neuen Container auf Basis von definierten Regeln starten. Durch das Deployment der Basis-Konfiguration wird das System automatisch konfiguriert, einschließlich Netzwerkeinstellungen, Sicherheitsrichtlinien usw. Automatisierte Workflows können Software-Deployment-Aufgaben durchführen, von der Installation von Anwendungen bis zur Konfiguration von Diensten. Die neue Instanz wird in den Loadbalancer aufgenommen, um den eingehenden Datenverkehr gleichmäßig auf die verschiedenen Server zu verteilen und die Verfügbarkeit zu maximieren.

### 12.3 Warum automatisiertes Deployment/Konfigurationsmanagement?



## Warum automatisiertes Deployment/ Konfigurationsmanagement?

- Wann gibt es Deployments?
- Wann werden Konfigurationen überprüft?
- manuelle Deployments und Konfiguration
- automatisierte Deployments und Konfiguration

Als Werkzeug für automatisiertes Deployment bietet Ansible viele Vorteile gegenüber dem manuellen Vorgehen. Mit Ansible wird der Prozess für Deployments nicht nur transparenter und strukturierter, da es eine einheitliche Form für die verschiedenen Prozessabschnitte gibt, sondern die Ergebnisse werden reproduzierbar. Somit werden viele Fehlerquellen von Anfang an reduziert und eine Fehleranalyse wird deutlich übersichtlicher. Zudem können die verschiedenen Anforderungen an Deployment-Prozesse durch einige Anpassungen alle mit Ansible und seinen Werkzeugen ausgeführt werden, und die einzelnen Komponenten dabei kombiniert und wiederverwendet.



## Wann gibt es Deployments?

- Testumgebungen
- Kundenbestellungen
- Upgrades
- Scale Out
- Restore fehlerhaft

Das Deployment ist der Gesamtprozess des automatisierten Bereitstellens von Software und Anwendungen. Das Deployment soll sowohl zeitsparsam als auch reproduzierbar und strukturiert sein. In jedem Schritt soll sichergestellt sein, dass die notwendigen Voraussetzungen gegeben sind und nicht unvollständig oder fehlerhaft ausgerollt wird.

Mögliche „Anlässe“ für Deployments sind beispielsweise das Ausrollen von Testumgebungen, die es Entwicklern ermöglichen, neue Funktionen und Änderungen zu testen, bevor sie in die Produktionsumgebung übernommen werden. Bestellen Kunden bestimmte Features oder Anwendungen, können diese einfach mit Deployments bereitgestellt werden. Auch für das Upgraden von Software bietet das automatisierte Deployen eine zeitsparende und zuverlässige Methode. Im Falle hoher Last können schnell zusätzliche Instanzen bereitgestellt werden (*Scale Out*). Auch bei einem fehlerhaften Restore ist schnelle Abhilfe möglich.

# Wann werden Konfigurationen überprüft?

- bei der Einrichtung
- für Audits
- im Fehlerfall

Konfigurationen können in Ansible zu verschiedenen Zeitpunkten überprüft werden, um die Konsistenz und Integrität der Systeme sicherzustellen und mögliche Fehler zu identifizieren.

Gerade bei der Einrichtung einer Software oder Anwendung kommt es darauf an, dass eine Konfiguration von Anfang an fehlerfrei ist. Ansible kann hier sicherstellen, dass nur die Systeme gestartet und bereitgestellt werden, die auch richtig konfiguriert wurden. Führt ihr Unternehmen Audits durch und überprüft, ob die eigenen Systeme bestimmten Compliance-Regeln entsprechen, kann Ansible hierbei auch eine zuverlässige Hilfestellung geben.

Bei Fehlermeldungen durch ein automatisiertes Monitoring oder durch Rückmeldung von Kunden oder Mitarbeitern ermöglicht Ansible, dass durch etablierte Prozesse schnell gehandelt und die Konfiguration überprüft werden kann, um z. B. eine Hochverfügbarkeit zu gewährleisten.



# Manuelle Deployments und Konfiguration

- langweilig, aufwändig und fehleranfällig
- schwer nachvollziehbar
- fehlende Dokumentation
- langsam
- nicht einfach reproduzierbar

Das manuelle Deployen und Konfigurieren kann dazu führen, dass während der Umsetzung durch stupides Arbeiten Fehler entstehen. Es ist zudem schwerer nachvollziehbar, an welchem Punkt ein Fehler passiert ist, oder ob alle notwendigen Schritte korrekt und vollständig abgeschlossen wurden.

Werden Deployments von verschiedenen Personen durchgeführt, fehlt mit der Automatisierung meist auch eine ausführliche Dokumentation. Für neue Mitarbeitende ist es daher oft schwieriger, in kurzer Zeit in den Workflow des Deployments einzusteigen. Je größer die Zahl der Deployments ist, desto bedeutender wird auch der Faktor Zeit. Je repetitiver Schritte sind, desto stärker macht sich das Fehlen einer Automatisierung bemerkbar.

Ein manuelles Deployment kann zwar beim ersten Mal zu einem befriedigenden Ergebnis führen, aber da die Schritte oft nicht einfach reproduzierbar sind, können Sie nicht davon ausgehen, dass Sie bei weiteren Deployments das gleiche Ergebnis erhalten.

Manuelle Deployments sind ineffizient und fehleranfällig. Automatisierungstools wie Ansible können dazu beitragen, Deployment-Prozesse zu standardisieren und Probleme zu minimieren.

# Automatisierte Deployments und Konfiguration

- einfach und sehr schnell reproduzierbar
- alle Änderungen nachvollziehbar
- Kommentare im Code
- Reporting um die Infrastruktur zu überwachen
- es ist sichergestellt, dass Systeme dem gewünschten Stand entsprechen

Automatisierte Deployments und Konfiguration bieten eine Vielzahl von Vorteilen gegenüber manuellen Prozessen:

- die Deployments und Konfigurationen können schnell reproduziert werden, was manuell nur mit deutlich höherem Zeitaufwand durchführbar wäre
- über die Konfiguration der Skripte und Logging kann nachvollzogen werden, wie und was an Änderungen durchgeführt wurde
- durch das Kommentieren der Skripte können Sie dokumentieren, mit welchen Zielen und auf welche Art bestimmte Aktionen durchgeführt werden
- das Reporting bietet Ihnen eine gute Möglichkeit, Einblicke über den Zustand der Infrastruktur zu bekommen und, wenn nötig, frühzeitig Schritte zu unternehmen
- es wird sichergestellt, dass die Systeme immer sicher und stabil sind, sowie bestimmten Compliance-Regeln entsprechen

## 12.4 Weitere Tools im Baukasten



# Weitere Tools im Baukasten

- Ansible Lint
- Ansible Galaxy
- Ansible Automation Platform

Für Ansible existieren weitere Werkzeuge im Baukasten, die bei der Verwaltung und Konfiguration helfen:

- *Ansible Lint* ist ein Kommandozeilenwerkzeug, das der Überprüfung von Playbooks, Rollen und Kollektionen dient und sich an Ansible-User wendet. Sein Hauptziel ist es, erprobte Praktiken und Muster zu empfehlen, um häufige Fallstricke zu vermeiden, die leicht zu Fehlern führen oder den Code schwerer wartbar machen können.
- *Ansible Galaxy* ist ein zentraler Ort, um Playbooks, Rollen und Kollektion zu teilen, ähnlich einem Repository (<https://galaxy.ansible.com/>).
- Die *Ansible Automation Platform* ist aus *Ansible Tower* entstanden. Es ist eine End-to-End-Plattform, die Ressourcen zur Erstellung, Verwaltung und Skalierung umfasst.

## 12.5 Alternativen zu Ansible



### Alternativen zu Ansible

- Welche Alternativen gibt es zu Ansible?
- Welche Tools kann man gut mit Ansible zusammen nutzen?

Neben Ansible gibt es weitere Werkzeuge, die ähnliche Aufgaben erfüllen. Genauso gibt es Tools, die gut in Kombination mit Ansible nutzbar sind. Diese werden auf den folgenden Seiten kurz vorgestellt.



## Alternativen zu Ansible

**CFEngine** alt eingesessen, akademisch, wenig Overhead  
(1993)

**Puppet** inspiriert durch CFEngine, eigene DSL (2005)

**Chef** inspiriert durch Puppet, noch näher an Ruby (2009)

**Salt** inspiriert durch Puppet/Chef, zusätzlich Remote Execution, YAML, Message Queue (2011)

Neben Ansible gibt es noch andere und ältere Softwaresysteme, die eine ähnliche Funktion bieten:

**CFEngine** wurde bereits 1993 von Mark Burgess programmiert und führte einige Innovationen bei der Verwaltung von Systemen ein. Es wird auch heute noch weiterentwickelt, ist sehr gut skalierbar, ressourcenschonend und schnell.

**Puppet** verwendet im Gegensatz zu Ansible ein agentenbasiertes Modell, das Agenten auf den Zielsystemen installiert. Dadurch bietet es auch eine bessere kontinuierliche Überwachung .

**Chef** wurde durch Puppet inspiriert, ermöglicht aber auch eine agentenlose Installation.

**Salt** zeichnet sich durch seine besonders hohe Geschwindigkeit und Skalierbarkeit sowie seine eingebaute Event-Bus-Funktionalität aus.

# Git

- verteilte Versionsverwaltung
- Änderungen nachvollziehbar
- Zusammenarbeit mehrerer Admins/Entwickler
- erleichtert Tests vor Rollout
- Verwaltung von Ansible Rollen & Playbooks

*Git* hat sich aktuell aufgrund seiner Flexibilität und Schnelligkeit als Standard bei Versionsverwaltungen durchgesetzt. Mit einer Versionsverwaltung können Sie Änderungen an Konfigurationen, Software und Dokumentationen mit einer Vielzahl an Kontributoren jederzeit nachvollziehen oder rückgängig machen. Eine verteilte Versionsverwaltung wie Git vereinfacht das Erstellen von individuellen Branches einzelner Beteiligter. Auf diese Weise können unterschiedliche Teams gleichzeitig an verschiedenen Teilen eines Projekts arbeiten.

Git bietet die Möglichkeit, Testumgebungen für Ansible-Rollen und Playbooks zu erstellen, indem verschiedene Branches oder Tags für bestimmte Umgebungen verwendet werden. Dies ermöglicht es, Änderungen in einer isolierten Umgebung zu testen, bevor sie in die Produktionsumgebung übernommen werden.



# Jenkins

- Continuous Integration Server
- automatisierte Builds von Software
- Ansible Plugin
- Playbooks/Ad-Hoc Kommandos als Build Step
- automatisches Ausrollen von Playbooks

Jenkins ist ein *Continuous Integration*-System, das die kontinuierliche Integration von Code-Änderungen, Tests und Builds in Softwareprojekten ermöglicht.

Entwickler können ihre Änderungen in ein zentrales Repository hochladen, Jenkins überwacht diese Änderungen und löst automatisch Build- und Testprozesse aus, um sicherzustellen, dass der Code fehlerfrei ist. Jenkins automatisiert so den Build-Prozess von Softwareprojekten. Dies kann auch für Ansible-Rollen und -Playbooks verwendet werden.

Jenkins bietet auch ein offizielles Ansible-Plugin, das die Integration von Ansible in Jenkins-Build-Pipelines ermöglicht. Entwickler können Ansible-Playbooks oder Ad-Hoc-Kommandos in Jenkins-Jobs als Teil des Build-Prozesses einrichten. Jenkins kann so konfiguriert werden, dass es automatisch Ansible-Playbooks auf Zielsystemen ausführt. Dies ist besonders nützlich für Continuous-Deployment-Szenarien, bei denen Änderungen in der Infrastruktur nach erfolgreichen Tests automatisch in die Produktionsumgebung übernommen werden.

Zusammengefasst ermöglicht Jenkins im Zusammenhang mit Ansible eine nahtlose Integration von Continuous Integration und Continuous Deployment (CI/CD) für die Infrastrukturautomatisierung. Dies erhöht die Zuverlässigkeit und Effizienz bei der Verwaltung von Konfigurationen und Automatisierungsskripten in einem Entwicklungsumfeld.

# The Foreman

- Lifecycle Management Server
- Deployment auf Hardware, VMs & Cloud
- Plugins für Puppet, Chef, Salt, Ansible
- Integration von Ansible in Deployment-Prozess
- mögliches Webinterface für Ansible

*The Foreman* ist ein Lifecycle-Management-Tool und Server, das in der Systemverwaltung und Infrastrukturautomatisierung zum Einsatz kommt. Mit The Foreman können Sie den gesamten Lebenszyklus von Systemen verwalten, einschließlich Provisioning, Konfiguration, Aktualisierung und Überwachung. Für das Deployment auf Hardware, VMs & Cloud ermöglicht es durch seine Unterstützung eine flexible, bedarfsorientierte Bereitstellung von Ressourcen. Das Tool bietet Plugins für eine Vielzahl von Konfigurationsmanagementtools, darunter auch für Ansible. Durch die Ansible-Integration können Ansible-Playbooks nahtlos in den Bereitstellungsprozess von The Foreman integriert werden. The Foreman bietet aber auch ein mögliches Webinterface für Ansible, das die Verwendung von Ansible erleichtert, z. B. durch Funktionen für die Erstellung, Verwaltung und Ausführung von Ansible-Playbooks.



## Docker & Podman

- Linux Container Engine
- Applikationen in eigener Laufzeitumgebung
- Ansible Rollen in Containern testen
- Container Images mit Ansible erstellen
- Container mit Ansible verwalten

*Docker* und *Podman* sind Linux-Container-Engines und ermöglichen es Entwicklern und Systemadministratoren, Containeranwendungen in einer eigenen Laufzeitumgebung auszuführen. Dies gewährleistet, dass Anwendungen in einer konsistenten Umgebung ausgeführt werden, unabhängig von den Bedingungen des Host-Systems. So können Sie Ansible-Rollen und Konfigurationen in Containern testen. Auch das Erstellen von Container Images ist mit Ansible möglich.

Podman bzw. Docker bietet eine nahtlose Integration mit Ansible, was es Benutzern ermöglicht, Ansible-Playbooks zu verwenden, um Container-Umgebungen und -Anwendungen effizient und reproduzierbar zu verwalten.

# Vagrant

- Werkzeug für die Verwaltung virtueller Umgebungen
- Infrastructure as Code
- VirtualBox, Hyper-V, VMware, Libvirt
- Ansible als Provisioner

*Vagrant* ist ein Werkzeug zur Verwaltung virtueller Umgebungen und spielt eine entscheidende Rolle im Bereich der Infrastrukturmehrheit und -verwaltung. Es ermöglicht Ihnen, virtuelle Maschinen (VMs) schnell und einfach bereitzustellen, zu konfigurieren und zu verwalten.

Ein wichtiger Aspekt bei der Nutzung von Vagrant ist die Idee von *Infrastructure as Code* (*IaC*), bei der die gesamte Infrastruktur mithilfe von Code definiert und verwaltet wird. Dies bedeutet, dass Sie Ihre VMs und deren Konfiguration in einem Text-basierten Format speichern können, was die Wiederholbarkeit, Skalierbarkeit und Zusammenarbeit erheblich verbessert.

Vagrant bietet Unterstützung für verschiedene Hypervisoren wie *VirtualBox*, *Hyper-V*, *VMware* und *Libvirt*. Ansible kann als Provisioner in die Vagrant-Umgebung integriert werden. So können Sie Ansible-Playbooks verwenden, um die Konfiguration und Bereitstellung Ihrer VMs zu automatisieren.



# Alternativen zum Ansible Automation Controller

- Semaphore
- Tensor
- Rundeck

Es gibt einige Alternativen zum *Ansible Automations Controller*, u. a. die folgenden:

**Semaphore** ist ein CI/CD-Tool, das nicht nur die Automatisierung von CI/CD-Pipelines ermöglicht, sondern auch die allgemeineren Aufgaben. Mit Semaphore können Sie Workflow-Pipelines erstellen und benutzerdefinierte Skripte in verschiedenen Programmiersprachen, einschließlich Ansible, verwenden. Es zeichnet sich durch eine benutzerfreundliche Oberfläche und umfangreiche Integrationsmöglichkeiten aus.

**Tensor** spezialisiert sich auf die Automatisierung von Aufgaben und Workflows. Mit einer YAML-basierten Syntax können Sie Aufgaben definieren und ausführen. Tensor bietet auch Rollen- und Rechteverwaltung sowie Integrationen mit verschiedenen Automatisierungstools und Plattformen.

**Rundeck** ist eine Plattform für Automatisierung und Job-Scheduling. Sie ermöglicht Ihnen die einfache Planung und Ausführung von Aufgaben, Skripten und Workflows über eine webbasierte Benutzeroberfläche. Rundeck unterstützt die Integration mit verschiedenen Automatisierungswerkzeugen, darunter Ansible, und erleichtert die Automatisierung von IT-Aufgaben und -Operationen.

## 13 Installation von Ansible



# Installation von Ansible

### 13.1 Zielsetzung



## Zielsetzung

Dieses Kapitel zeigt die Installation von Ansible

- über Distributionspakete
- via PIP

## 13.2 Installation über Distributionspakete



# Installation über Distributionspakete – RHEL

## RHEL: Installation mit yum

```
# subscription-manager repos --enable \
    ansible-2.9-for-rhel-8-x86_64-rpms
# yum install ansible
```

Bei RHEL 8 wird Ansible aus der „Ansible Engine Software Collection“ installiert. Dies hat den Vorteil, dass keine externen Repositories wie EPEL mehr genutzt werden müssen. Die Collections sind vollständig von Red Hat unterstützt und in Ihrem *Subscription Plan* enthalten.

Die Ansible Collection muss zunächst aktiviert werden, um anschließend Ansible installieren zu können:

```
# subscription-manager repos --enable \
    ansible-2.9-for-rhel-8-x86_64-rpms
# yum install ansible
```



# Installation über Distributionspakete – SLES 15

## SLES 15: Installation mit zypper

```
# SUSEConnect -p PackageHub/15.0/x86_64
Registered PackageHub 15.0 x86_64
To server: https://scc.suse.com
Successfully registered system.

# SUSEConnect -p sle-module-public-cloud/15.0/x86_64
Registered sle-module-public-cloud 15.0 x86_64
To server: https://scc.suse.com
Successfully registered system.

# zypper install ansible
```

In SLES 15 müssen für die Installation von Ansible zunächst zwei Module über das *SUSE Customer Center Registration Tool* SUSEConnect registriert werden. Ansible ist im Modul PackageHub enthalten. Das Modul sle-module-public-cloud enthält die benötigte Abhängigkeit python3-paramiko.

Mit dem Befehl SUSEConnect und der Option -p bzw. --product werden die Module für die entsprechende SLES-Version und Prozessorarchitektur hinzugefügt:

```
# SUSEConnect -p PackageHub/15.0/x86_64
Registered PackageHub 15.0 x86_64
To server: https://scc.suse.com
Successfully registered system.

# SUSEConnect -p sle-module-public-cloud/15.0/x86_64
Registered sle-module-public-cloud 15.0 x86_64
To server: https://scc.suse.com
Successfully registered system.
```

Anschließend installieren Sie Ansible mit zypper:

```
# zypper install ansible
```



## Installation über Distributionspakete – Ubuntu/Debian

### Debian/Ubuntu: Installation mit apt

```
$ sudo apt update  
$ sudo apt install ansible
```

### Spezielles Repository unter Ubuntu verwenden:

```
$ sudo apt update  
$ sudo apt install software-properties-common  
$ sudo add-apt-repository --yes --update ppa:ansible/ansible  
$ sudo apt install ansible
```

Die Installation von Ansible und der zugehörigen Dokumentation erfolgt unter Debian und Ubuntu mit apt:

```
$ sudo apt update  
$ sudo apt install ansible
```

Unter Ubuntu können Sie alternativ eine spezielle PPA (*Personal Package Archive*, Paketquelle) hinzufügen, in der aktuellere Releases bereitgestellt werden:

```
$ sudo apt update  
$ sudo apt install software-properties-common  
$ sudo add-apt-repository --yes --update ppa:ansible/ansible  
$ sudo apt install ansible
```



## Installation mit PIP

### Installation mit pip unter RHEL und SLES

```
# easy_install pip  
# pip install ansible
```

### Installation mit pip unter Ubuntu

```
# apt install python3-pip  
# pip3 install ansible
```

### Festlegen der Ansible-Version bei der Installation

```
# pip install 'ansible==<Versionsnummer>'
```

Distributionsunabhängig ist eine Installation von Ansible über den Python-Paketverwalter `pip` (*Pip Installs Packages*) anstelle aus den Repositories der jeweiligen Distribution möglich. Dies hat den Vorteil, dass über `pip` genau festgelegt werden kann, in welcher Version Ansible installiert werden soll.

Zunächst muss dafür `pip` installiert werden. Unter SLES und RHEL installieren Sie `pip` über das Python-Modul `easy_install`:

```
# easy_install pip
```

Bei Ubuntu wird `pip` über den Paketmanager `apt` installiert:

```
# apt install python3-pip
```

Ist `pip` vorhanden, kann darüber Ansible installiert werden:

```
# pip install ansible
```

Unter Ubuntu lautet der Befehl zur Installation `pip3`:

```
# pip3 install ansible
```

## 13 Installation von Ansible

Mit pip kann die zu installierende Version von Ansible festgelegt werden, sofern nicht die aktuellste verfügbare Version installiert werden soll. Geben Sie die Versionsnummer wie folgt an:

```
# pip install 'ansible==<Versionsnummer>'
```

also z. B.:

```
# pip install 'ansible==4.1'
```



# Installation mehrerer Ansible-Versionen in Virtual Python Environment

## Erstellen eines Virtual Python Environment:

```
# python3 -m virtualenv ansible3
```

## Ansible in einer virtuellen Python-Umgebung installieren:

```
# source ansible3/bin/activate
(ansible3) tux@sles: $ 
(ansible3) tux@sles: $ python3 -m pip \
install ansible-core==2.11.2
```

Persönliches Exemplar von [peter.wohlfarth@justiz.thueringen.de](mailto:peter.wohlfarth@justiz.thueringen.de)  
 Durch die Verwendung sogenannter *Virtual Python Environments* (virtuelle Python-Umgebungen) haben Sie die Möglichkeit, gleichzeitig mehrere unterschiedliche Ansible-Versionen auf einem System zu installieren.

Dafür richten Sie zunächst eine virtuelle Umgebung ein, wechseln anschließend in diese und installieren dann Ansible in der gewünschten Version.

Eine virtuelle Umgebung erstellen Sie wie folgt:

```
# mkdir ansible3
# python3 -m virtualenv ansible3
```

Legen Sie zunächst ein Verzeichnis für die neue Umgebung an (oder verwenden Sie ein bereits vorhandenes Verzeichnis). Anschließend richten Sie die Umgebung mittels des Python-Moduls `virtualenv` in dem Verzeichnis ein.

Nach Anlegen der Umgebung wechseln Sie in diese:

```
# source ansible3/bin/activate
(ansible3) tux@sles:~>
```

Durch den Wechsel in die virtuelle Umgebung verändert sich Ihr Prompt. In Klammern wird das Verzeichnis (hier `ansible3`) der virtuellen Umgebung angezeigt.

## 13 Installation von Ansible

Innerhalb der Umgebung installieren Sie mit pip die gewünschte Ansible-Version:

```
(ansible3) tux@sles:~> python3 -m pip install ansible-core==2.11.2
```

In der virtuellen Umgebung `ansible3` ist nun `ansible-core` in der Version 2.11.2 installiert und nutzbar. Durch Eingabe von `deactivate` verlassen Sie die virtuelle Umgebung:

```
(ansible3) tux@sles:~> deactivate  
tux@sles:~>
```

Um wieder in die virtuelle Umgebung zu wechseln, wiederholen Sie den `source <Umgebungsname>/bin/activate`-Befehl. Sie können eine beliebige Zahl von verschiedenen virtuellen Umgebungen anlegen und somit verschiedene Ansible-Versionen parallel installieren.

### 13.3 Aufgabenteil



## Aufgabenteil – Ansible-Installation

Installieren Sie Ansible auf dem Control-Host mittels der Pakete aus dem Debian-Repository.

Installieren Sie Ansible auf dem Control-Host mittels der Pakete aus dem Debian-Repository.

## Musterlösung

**Installieren Sie Ansible auf dem Control-Host mittels der Pakete aus dem Debian-Repository:**

Update des Paket-Caches:

```
$ sudo apt update
Hit:1 http://deb.debian.org/debian bookworm InRelease
Hit:2 http://deb.debian.org/debian bookworm-updates InRelease
Hit:3 http://security.debian.org/debian-security bookworm-security
      ↗ InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
All packages are up to date.
```

Installation des Pakets ansible-core:

```
$ sudo apt install ansible
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  ansible-core ieee-data python3-distutils python3-dnspython
    ↗ python3-lib2to3 python3-netaddr python3-packaging
    ↗ python3-pycryptodome python3-resolverlib
Suggested packages:
  cowsay sshpass python3-sniffio python3-trio python3-aioquic ipython3
    ↗ python-netaddr-docs
Recommended packages:
  python3-argcomplete python3-jmespath python3-kerberos
    ↗ python3-libcloud python3-selinux python3-winrm
    ↗ python3-xmldict python3-h2 python3-httplib
    ↗ python3-requests-toolbelt
The following NEW packages will be installed:
  ansible ansible-core ieee-data python3-distutils python3-dnspython
    ↗ python3-lib2to3 python3-netaddr python3-packaging
    ↗ python3-pycryptodome python3-resolverlib
0 upgraded, 10 newly installed, 0 to remove and 0 not upgraded.
Need to get 21.3 MB of archives.
After this operation, 238 MB of additional disk space will be used.
Do you want to continue? [Y/n]
Get:1 http://deb.debian.org/debian bookworm/main amd64
      ↗ python3-packaging all 23.0-1 [32.5 kB]
Get:2 http://deb.debian.org/debian bookworm/main amd64
      ↗ python3-resolverlib all 0.9.0-2 [27.3 kB]
Get:3 http://deb.debian.org/debian bookworm/main amd64
      ↗ python3-pycryptodome amd64 3.11.0+dfsg1-4 [1011 kB]
Get:4 http://deb.debian.org/debian bookworm/main amd64 python3-lib2to3
      ↗ all 3.11.2-3 [76.3 kB]
Get:5 http://deb.debian.org/debian bookworm/main amd64
      ↗ python3-distutils all 3.11.2-3 [131 kB]
Get:6 http://deb.debian.org/debian bookworm/main amd64
      ↗ python3-dnspython all 2.3.0-1 [152 kB]
```

```

Get:7 http://deb.debian.org/debian bookworm/main amd64 ieee-data all
  ↳ 20220827.1 [2029 kB]
Get:8 http://deb.debian.org/debian bookworm/main amd64 python3-netaddr
  ↳ all 0.8.0-2 [295 kB]
Get:9 http://deb.debian.org/debian bookworm/main amd64 ansible-core
  ↳ all 2.14.3-1 [1273 kB]
Get:10 http://deb.debian.org/debian bookworm/main amd64 ansible all
   ↳ 7.3.0+dfsg-1 [16.3 MB]
Fetched 21.3 MB in 0s (135 MB/s)
Selecting previously unselected package python3-packaging.
(Reading database ... 22182 files and directories currently installed.)
Preparing to unpack .../0-python3-packaging_23.0-1_all.deb ...
Unpacking python3-packaging (23.0-1) ...
Selecting previously unselected package python3-resolvelib.
Preparing to unpack .../1-python3-resolvelib_0.9.0-2_all.deb ...
Unpacking python3-resolvelib (0.9.0-2) ...
Selecting previously unselected package python3-pycryptodome.
Preparing to unpack
   ↳ .../2-python3-pycryptodome_3.11.0+dfsg1-4_amd64.deb ...
Unpacking python3-pycryptodome (3.11.0+dfsg1-4) ...
Selecting previously unselected package python3-lib2to3.
Preparing to unpack .../3-python3-lib2to3_3.11.2-3_all.deb ...
Unpacking python3-lib2to3 (3.11.2-3) ...
Selecting previously unselected package python3-distutils.
Preparing to unpack .../4-python3-distutils_3.11.2-3_all.deb ...
Unpacking python3-distutils (3.11.2-3) ...
Selecting previously unselected package python3-dnspython.
Preparing to unpack .../5-python3-dnspython_2.3.0-1_all.deb ...
Unpacking python3-dnspython (2.3.0-1) ...
Selecting previously unselected package ieee-data.
Preparing to unpack .../6-ieee-data_20220827.1_all.deb ...
Unpacking ieee-data (20220827.1) ...
Selecting previously unselected package python3-netaddr.
Preparing to unpack .../7-python3-netaddr_0.8.0-2_all.deb ...
Unpacking python3-netaddr (0.8.0-2) ...
Selecting previously unselected package ansible-core.
Preparing to unpack .../8-ansible-core_2.14.3-1_all.deb ...
Unpacking ansible-core (2.14.3-1) ...
Selecting previously unselected package ansible.
Preparing to unpack .../9-ansible_7.3.0+dfsg-1_all.deb ...
Unpacking ansible (7.3.0+dfsg-1) ...
Setting up python3-pycryptodome (3.11.0+dfsg1-4) ...
Setting up python3-resolvelib (0.9.0-2) ...
Setting up python3-packaging (23.0-1) ...
Setting up ieee-data (20220827.1) ...
Setting up python3-dnspython (2.3.0-1) ...
Setting up python3-lib2to3 (3.11.2-3) ...
Setting up python3-distutils (3.11.2-3) ...
Setting up python3-netaddr (0.8.0-2) ...
Setting up ansible-core (2.14.3-1) ...
Setting up ansible (7.3.0+dfsg-1) ...
Processing triggers for man-db (2.11.2-2) ...

```

## 14 Ansible Ad-Hoc Befehle



# Ansible Ad-Hoc Befehle

## 14.1 Zielsetzung



# Zielsetzung

Dieses Kapitel bietet eine Einführung in die Grundlagen von Ansible anhand von Ad-Hoc-Befehle:

- Grundlagen der Umgebung
- Inventory
- Module
- Collections

## 14.2 Ad-Hoc-Befehle



# Ad-Hoc-Befehle

- Ad-Hoc-Befehle: Änderungen einmalig und direkt von der Kommandozeile aus auf Zielsystemen umsetzen
- Ausführung einzelner Module auf Zielhost
- einmalige Aktionen, z. B.:
  - Notfall-Patches ausrollen
  - Nameserver ändern
- Informationsgewinnung, z. B.:
  - Hardware
  - OS Releases

Mit den sog. „*Ad-hoc*“-Befehlen bietet Ansible die Möglichkeit, Änderungen einmalig und direkt von der Kommandozeile aus auf den Zielsystemen umzusetzen.

In der Regel wird auf dieses Vorgehen zugunsten von umfangreicheren Werkzeugen wie Playbooks verzichtet, die jedoch im Wesentlichen dieselben Techniken nutzen. Manchmal kann es auch vorteilhaft sein, einzelne Änderungen an einem Zielsystem vorzunehmen, ohne dass dabei umfangreiche Werkzeuge genutzt werden müssen.

Ad-Hoc-Befehle werden direkt auf der Kommandozeile ausgeführt und können ein einzelnes oder mehrere Systeme auf einmal ansprechen. Ausgehend von einem *Ansible Controller* können so Kommandos ausgeführt, Informationen gesammelt oder Software auf den Zielsystemen gepflegt werden. Im Folgenden wird anhand der Ad-Hoc-Befehle der grundlegende Aufbau sowie die Funktionsweise von Ansible vermittelt.



## Ad-hoc-Befehle bei Ansible

- direkt einmalig vom Controller aus Kommandos auf allen Zielsystemen ausführen

Ad-hoc-Befehl mit Modul `ansible.builtin.setup` zur Informationsgewinnung:

```
# ansible all -i code/inventory -m ansible.builtin.setup
```

Ad-hoc-Befehl mit `become`-Option:

```
# ansible all -i code inventory -b \
-m ansible.builtin.package -a "name=tmux state=present"
```

Mit *Ad-hoc*-Befehlen können Sie bei Ansible direkt einmalig vom Controller aus Kommandos auf allen Zielsystemen ausführen lassen. Dazu gibt es das CLI-Tool `ansible`. `ansible` verwendet mit dem Schalter `-m` die gleichen Ansible-Module (siehe Definition Ansible-Modul) wie die Playbooks. Gegebenenfalls werden die Argumente mit dem Schalter `-a` für das Modul mit übergeben. Sinnvollerweise werden die Argumente mit Anführungszeichen (" oder ') vor der Bash versteckt.

Meist werden Ad-hoc-Befehle nur mit dem Modul `ansible.builtin.setup` zur Informationsgewinnung ausgeführt:

```
# ansible all -i code/inventory -m ansible.builtin.setup
```

Ansible führt das Modul `Ansible` führt das Modul `ansible.builtin.setup` grundsätzlich beim Start jedes Playbooks aus. Es füllt die Dictionary-Struktur `ansible_facts` mit Informationen zu jedem einzelnen Zielsystem. Jeder Teil der `ansible-facts` kann danach ausgewertet werden. So können Sie beispielsweise über den Ansible-Fact `ansible_os_family` die Linux-Distribution ermitteln und Tasks lassen sich abhängig von einer bestimmten Linux-Distribution ausführen.

Für das Debugging komplexerer Szenarien können sie auch gut verwendet werden, da auf der Kommandozeile (Test-)Variablen-Belegungen mitgegeben werden können. Beim Debugging/Testen von Playbooks ist es oft sinnvoll, den Code gleich ins Playbook zu schreiben und es dann durchführen zu lassen. Bei den „falschen“ Befehlen stellt ein erneuter Playbook-Durchlauf ja wieder den richtigen Zustand her.

## 14 Ansible Ad-Hoc Befehle

Die Module `ansible.builtin.ping` und `ansible.builtin.setup` benötigen keine root-Rechte auf dem Zielsystem.

Für eine Software-Installation mit `ansible.builtin.package` sind root-Rechte dagegen erforderlich.

Die eingangs erwähnte `become`-Option wird bei Ad-hoc-Befehlen mit dem Schalter `-b` genutzt:

```
# ansible all -i code inventory -b -m ansible.builtin.package -a  
    ↳ "name=tmux state=present"
```



## Ad-Hoc-Befehle 1/3

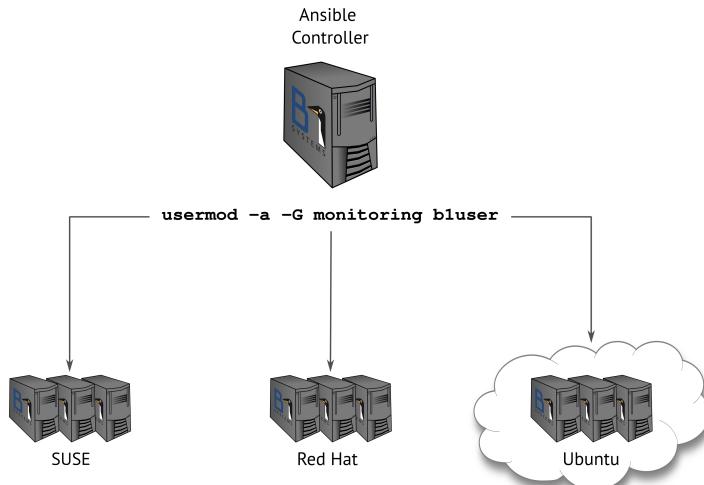


Abbildung: Kommandos auf vielen Hosts ausführen

Mit den Ad-Hoc-Befehle können Befehle auf mehreren Systemen oder einer ganzen definierten Ziellandschaft gleichzeitig ausgeführt werden.

Der Aufbau einer Ansible-Umgebung setzt voraus, dass es einen *Controller* gibt, auf dem Ansible installiert ist. Auf den Zielsystemen muss lediglich Python vorhanden sein. Für die gesamte Umgebung ist nur ein einziger Controller notwendig.

## Ad-Hoc-Befehle 2/3

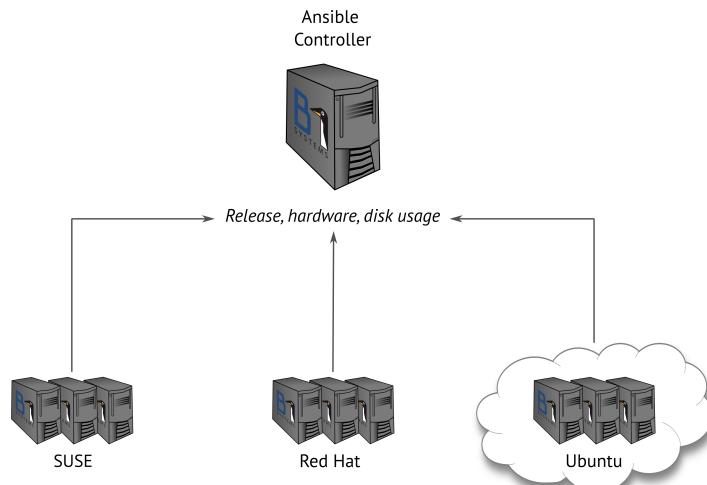


Abbildung: Informationen von Hosts abrufen

Ebenso können Informationen über die Systeme gesammelt und an den Controller zurückgesendet werden.



## Ad-Hoc-Befehle 3/3

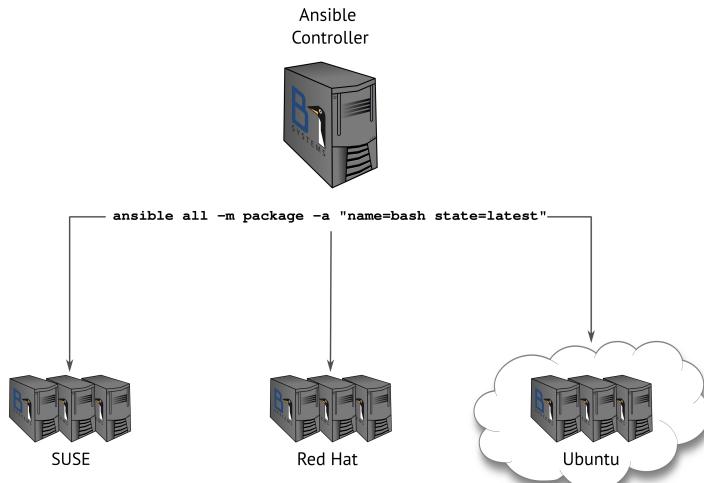


Abbildung: Softwarepakete auf vielen Systemen aktualisieren

Durch die Möglichkeit, Befehle auf vielen Systemen gleichzeitig auszuführen, können beispielsweise Updates auf allen Systemen gleichzeitig angestoßen oder eingespielt werden. Dafür ist lediglich das Bedienen des Controllers nötig. Die Zielsysteme können ihren Status an den Controller zurückschicken.



## Ad-Hoc-Befehle – Aufruf

### Aufruf von Ad-Hoc-Befehle

```
$ ansible <ZIELHOST> [OPTIONS] -m <MODULE> -a <MODULARGS>
```

Die einfachste Art, mit `ansible` einen Befehl auszuführen, sind Ad-Hoc-Befehle. Bei dieser Art von Befehl wird direkt auf der Kommandozeile eine Aktion definiert und auf einem oder mehreren Zielsystemen ausgeführt. Der grundlegende Aufbau eines Ad-Hoc Befehls gestaltet sich wie folgt:

```
$ ansible <ZIELHOST> [OPTIONEN] -m <MODULE> -a <MODULARGUMENTE>
```

Zunächst muss der Name des Zielsystems angegeben werden, auf dem `ansible` die Aktion ausführen soll. Es folgen Optionen und die Aktion selber in Form eines Moduls, wie z. B. `ansible.builtin.ping`.

### 14.2.1 Aufgabenteil



## Aufgabenteil – Ad-Hoc-Befehle: Aufruf

Führen Sie ein Ad-Hoc Kommando mit dem Zielhost `ubuntu1` und dem Modul `ansible.builtin.ping` aus.

Führen Sie ein Ad-Hoc Kommando mit dem Zielhost `ubuntu1` und dem Modul `ansible.builtin.ping` aus.

## Musterlösung

**Führen Sie ein Ad-Hoc Kommando mit dem Zielhost `ubuntul` und dem Modul `ansible.builtin.ping` aus:**

```
$ ansible ubuntul -m ansible.builtin.ping
[WARNING]: Could not match supplied host pattern, ignoring: ubuntul
[WARNING]: No hosts matched, nothing to do
```

Das Ausführen des Befehls schlägt fehl, da das Zielsystem nicht im *Inventory* von Ansible eingetragen ist.

### 14.3 Inventory



## Inventory

- Liste der zu verwaltenden Systeme
- *Managed Nodes* müssen im Inventory hinterlegt sein
- kann Variablen enthalten
- verschiedene Formate (INI ist Standard)
- /etc/ansible/hosts oder mit Option -i
- Authentifizierung (SSH) beachten

---

B1 Systems GmbH      Terraform & Ansible Grundlagen      135 / 279

Im *Inventory* müssen alle Systeme eingetragen werden, die mit Ansible verwaltet werden sollen. Wird `ansible` mit einem Zielhost aufgerufen, der nicht im Inventory vorhanden ist, erscheint die folgende Fehlermeldung:

```
$ ansible ubuntul -m ansible.builtin.ping
[WARNING]: Could not match supplied host pattern, ignoring:
[WARNING]: No hosts matched, nothing to do
```

Das Inventory ist eine einfache Textdatei, in der die Systeme aufgelistet sind, z. B.:

```
$ cat inventory
```

```
172.20.1.121
```

So müsste der Eintrag aussehen, wenn keine DNS-Auflösung vorhanden ist. Kann der Hostname von einem DNS-Server aufgelöst werden, genügt es, den Hostnamen in das Inventory einzutragen:

```
ubuntul
```

Ansible erwartet das Inventory standardmäßig in der Datei `/etc/ansible/hosts`. Eine abweichende Datei geben Sie mittels `-i` an.

## 14 Ansible Ad-Hoc Befehle

Der Befehl kann nun unter Angabe des Inventory mit der Option `-i` erneut ausgeführt werden:

```
$ ansible ubuntul -i inventory -m ansible.builtin.ping
ubuntul | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: Permission denied
        ↪ (publickey,password).",
    "unreachable": true
}
```

Diesmal scheitert die Ausführung des Moduls an der Authentifizierung. Da `ansible` eine SSH-Verbindung aufbaut und keine automatischen Authentifizierungsmechanismen wie SSH-Keys eingerichtet sind, muss `ansible` hier mit der Option `-k` bzw. `--ask-pass` dazu aufgefordert werden, nach einem Passwort zu fragen:

```
$ ansible ubuntul -i inventory -k -m ansible.builtin.ping
SSH password:
```

Nach der Eingabe des Passworts erscheint nun:

```
ubuntul | FAILED! => {
    "msg": "Using a SSH password instead of a key is not possible
        ↪ because Host Key checking is enabled and sshpass does not
        ↪ support this. Please add this host's fingerprint to your
        ↪ known_hosts file to manage this host."
}
```

Also muss noch der SSH-Key des Zielhosts in die `~/.ssh/known_hosts` eingetragen werden:

```
$ ssh-keyscan ubuntul >> ~/.ssh/known_hosts
```

Nun kann der Ad-Hoc Befehl ausgeführt und das Passwort eingegeben werden und es erscheint:

```
$ ansible ubuntul -i inventory -k -m ansible.builtin.ping
SSH password:
ubuntul | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Der Einfachheit halber können Sie alle Hosts auf einmal mit folgendem Befehl in die `known_hosts` eintragen:

```
$ ssh-keyscan -f inventory >> ~/.ssh/known_hosts
```

### 14.3.1 Aufgabenteil



## Aufgabenteil – Inventory

- ① Legen Sie sich ein Arbeitsverzeichnis /home/tux/ansible an.
- ② Tragen Sie alle verfügbaren Zielsysteme in das Inventory /home/tux/ansible/inventory ein.
- ③ Führen Sie den folgenden Ad-Hoc Befehl für alle im Inventory eingetragenen Zielsysteme aus:  
`$ ansible all -i inventory -k -m  
ansible.builtin.ping`

1. Legen Sie sich ein Arbeitsverzeichnis /home/tux/ansible an.
2. Tragen Sie alle verfügbaren Zielsysteme in das Inventory /home/tux/ansible/inventory ein.
3. Führen Sie den folgenden Ad-Hoc Befehl für alle im Inventory eingetragenen Zielsysteme aus:  
`$ ansible all -i inventory -k -m ansible.builtin.ping`

## Musterlösung

1. Legen Sie sich ein Arbeitsverzeichnis `/home/tux/ansible` an:

```
$ mkdir ansible
```

2. Tragen Sie alle verfügbaren Zielsysteme in das Inventory `/home/tux/ansible/inventory` ein:

Der Inhalt der Inventory-Datei sollte wie folgt aussehen:

```
$ cat inventory
ubuntul
ubuntu2
centos1
centos2
suse1
suse2
```

3. Führen Sie den folgenden Ad-Hoc Befehl für alle im Inventory eingetragenen Zielsysteme aus:

```
$ ansible all -i inventory -k -m ansible.builtin.ping:
```

Mit dem Parameter `all` führen Sie einen Ad-Hoc Befehl für alle im Inventory enthaltenen Zielsysteme aus:

```
$ ansible all -i inventory -k -m ansible.builtin.ping
SSH password:
ubuntul | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
centos1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
centos2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
suse2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
suse1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
ubuntu2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

### 14.3.2 SSH-Keys



## Authentifizierung per SSH-Key

### SSH-Key erzeugen mit ssh-keygen:

```
$ ssh-keygen
Generating public/private rsa key pair.
```

### SSH-Key auf dem Zielsystem platzieren:

```
$ ssh-copy-id <Zielsystem>
```

Um zu vermeiden, dass Sie bei jedem Ausführen eines Befehls ein Passwort eingeben müssen, können Sie einen SSH-Key ohne Passphrase erzeugen und auf dem Zielsystem hinterlegen. Ist ein solcher Schlüssel vorhanden, können Sie mit Ansible Aktionen ohne Passwort-Eingabe ausführen.

### SSH-Key erzeugen mit ssh-keygen:

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/tux/.ssh/id_rsa):
Created directory '/home/tux/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/tux/.ssh/id_rsa.
Your public key has been saved in /home/tux/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:1XHnD7IdbYUHSwU/rCajyN/fWOS2MNWvHpmO0uegvgE
    ↪ tux@training.b1-systems.de
The key's randomart image is:
+---[RSA 3072]---+
|          . . ==o|
|          + +o+o|
```

## 14 Ansible Ad-Hoc Befehle

```
|          o . =++ |
|          . +.=o |
|          E  o..oo |
|          . ... + +o. |
|          o  ....o++. |
|          . oo.+Oo. |
|          o++o*=o  |
+--- [ SHA256 ] ---+
```

Wichtig ist, dass Sie keine Passphrase festlegen!

SSH-Key auf dem Zielsystem platzieren:

```
$ ssh-copy-id <Zielsystem>
```

### 14.3.3 Aufgabenteil



## Aufgabenteil – SSH-Key

- ① Erzeugen Sie einen SSH-Key.
- ② Kopieren Sie den erzeugten Key auf das System ubuntu1.

1. Erzeugen Sie einen SSH-Key.
2. Kopieren Sie den erzeugten Key auf das System ubuntu1.

## Musterlösung

### 1. Erzeugen Sie einen SSH-Key:

Sie erzeugen einen SSH-Key mit dem Programm ssh-keygen:

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/tux/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/tux/.ssh/id_rsa.
Your public key has been saved in /home/tux/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:9t9dd4AZZq+IRdHAAj5qVnMYeElOgoa3MCJA8HWxv9Y tux@ansible
The key's randomart image is:
+---[RSA 2048]---+
| * . . . o . oo   |
| o . + . . . o ... |
| . . . = . . * o . + |
| . . . B . o =     |
| + S o . o o       |
| . * * . . . .     |
| o ++ E . . +     |
| + . . . . . +     |
| o . . . . . |     |
+---[SHA256]---
```

### 2. Kopieren Sie den erzeugten Key auf das System **ubuntul**:

Mit ssh-copy-id kopieren Sie den erzeugten Schlüssel auf das Zielsystem:

```
# ssh-copy-id ubuntul
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed:
    ↪ "/home/tux/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new
    ↪ key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if
    ↪ you are prompted now it is to install the new keys
tux@ubuntul's password:
```

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'ubuntul'"  
and check to make sure that only the key(s) you wanted were added.

#### 14.3.4 Gruppen im Inventory



## Hosts im Inventory verschiedenen Gruppen zuteilen

Hosts können im Inventory in verschiedene Gruppen eingeteilt werden

- Formatierung nach INI-Standard
- Host kann mehreren verschiedenen Gruppen zugeordnet sein

Beispiel für eine Gruppe im Inventory:

```
[Gruppe1]
System1
System2
```

Aufruf der Gruppe mit Ansible:

```
$ ansible Gruppe1 -i inventory -k -m ansible.builtin.ping
```

Die Inventory-Dateien können u. a. im INI-Format angelegt werden. Daher sind Einträge im Inventory nach folgendem Schema gruppierbar:

```
[Gruppe1]
ubuntul
ubuntu2
```

```
[Gruppe2]
suse1
suse2
```

Dabei können einzelne Einträge auch verschiedenen Gruppen zugeordnet sein:

```
[Gruppe1]
ubuntul
ubuntu2
```

```
[Gruppe2]
ubuntul
suse2
```

## 14 Ansible Ad-Hoc Befehle

Anschließend können Sie Ad-Hoc Befehle für Gruppen ausführen:

```
$ ansible Gruppe1 -i inventory -k -m ansible.builtin.ping
```

### 14.3.5 Aufgabenteil



## Aufgabenteil – Gruppen im Inventory

Gruppieren Sie alle verfügbaren Zielsysteme im Inventory nach ihrem Betriebssystem.

Gruppieren Sie alle verfügbaren Zielsysteme im Inventory nach ihrem Betriebssystem.

## Musterlösung

**Gruppieren Sie alle verfügbaren Zielsysteme im Inventory nach ihrem Betriebssystem:**

Die Zielsysteme in Gruppen eintragen (/home/tux/ansible/inventory):

```
[ubuntu]
```

```
ubuntu1
```

```
ubuntu2
```

```
[centos]
```

```
centos1
```

```
centos2
```

```
[suse]
```

```
suse1
```

```
suse2
```

## 14.4 Targeting-Optionen für Ad-Hoc Befehle



# Targeting-Optionen für Ad-Hoc Befehle 1/2

Mehrere Zielsysteme ansprechen:

```
$ ansible ubuntu1,suse1 -i inventory -m ansible.builtin.ping
```

Mehrere Gruppen ansprechen:

```
$ ansible group1,group2 -i inventory -m ansible.builtin.ping
```

Gruppen und einzelne Hosts kombinieren:

```
$ ansible group1,suse2 -i inventory -m ansible.builtin.ping
```

Neben der Möglichkeit, einen Ad-Hoc Befehl an jeweils *einen* Host oder *eine* Gruppe zu adressieren, gibt es beim *Targeting* auch die Möglichkeit, Hosts und Gruppen zu kombinieren.

Mehrere Zielsysteme ansprechen:

```
$ ansible ubuntu1,suse1,... -i inventory -m ansible.builtin.ping
```

Mehrere Gruppen ansprechen:

```
$ ansible Gruppe1,Gruppe2,... -i inventory -m ansible.builtin.ping
```

Gruppen und einzelne Hosts kombinieren:

```
$ ansible Gruppe1,ubuntu1,... -i inventory -m ansible.builtin.ping
```

## Targeting-Optionen für Ad-Hoc Befehle 2/2

### Einzelne Systeme oder Gruppen ausschließen:

```
$ ansible all,'!Gruppe2','!ubuntu1',... -i inventory \
-m ansible.builtin.ping
```

### Einzelne Systeme oder Gruppen auswählen:

```
$ ansible all -i inventory -m ansible.builtin.ping \
--limit=centos,suse1
```

### Zielsystem ohne Inventory-Eintrag ansprechen:

```
$ ansible all -i 172.20.1.121, -m ansible.builtin.ping
```

Mit der Option `all` sprechen Sie alle im Inventory eingetragenen Zielsysteme an:

```
$ ansible all -i inventory -m ansible.builtin.ping
```

Es können auch einzelne Systeme oder Gruppen mit einem „!“ vom Aufruf ausgeschlossen werden:

```
$ ansible all,'!ubuntu1' -i /home/tux/ansible/inventory -m
    ↪ ansible.builtin.ping
```

Dies würde alle im Inventory eingetragenen Systeme ansprechen, jedoch das System `ubuntu1` ausschließen.

```
$ ansible all,'!Gruppe1' -i /home/tux/ansible/inventory -m
    ↪ ansible.builtin.ping
```

Dies würde alle im Inventory eingetragenen Systeme ansprechen, jedoch die Gruppe `Gruppe1` ausschließen.

```
$ ansible all -i inventory -m ansible.builtin.ping --limit=centos,suse1
```

Mit der Option `--limit` kann der Aufruf von Ansible auf einzelne Systeme der aufgerufenen Gruppe beschränkt werden.

## Targeting ohne Inventory

Neben den Targeting-Methoden über eine Inventory-Datei können Sie dem ansible-Befehl auch direkt eine IP-Adresse oder Domain übergeben, die nicht in einem Inventory eingetragen wurde.:

```
$ ansible all -i 172.20.1.121, -m ansible.builtin.ping
```

Dabei ist zu beachten, dass als Zielsystem `all` angegeben werden muss und der Option `-i` der Host im Format „Liste“ übergeben wird. Durch das Komma (`172.20.1.121,`) am Ende der IP-Adresse wird diese zum ersten Eintrag einer Liste deklariert. Dies ist notwendig, da die Option `-i` ein als Liste vorliegendes Inventory erwartet. Die Anzahl der Einträge in dieser Liste kann beliebig erweitert werden, sodass das Ad-Hoc Inventory als kommasseparierte Liste (CSV) auch mehrere Hosts enthalten kann. Wichtig ist lediglich, dass die Liste mit einem Komma beendet wird.

 **Tipp** *Targeting in Ansible*  
Die hier gezeigten Targeting-Optionen gelten auch in Kombination mit dem im folgenden Kapitel vorgestellten Befehl `ansible-playbook`.

## 14.5 Ansible-Module



# Ansible-Module 1/2

- führen spezielle Aufgaben aus
- sind seit Ansible 3.0 (`ansible-core 2.10`) in *Collections* unterteilt
- unterstützen Argumente
- Ausführung Ad-Hoc oder als Teil eines Playbooks
- Beispiele: `ansible.builtin.package`,  
`ansible.builtin.setup`

Module sind Skripte, die auf dem Zielsystem ausgeführt werden. Mit Hilfe der Module verwaltet Ansible die Zielsysteme und nimmt Änderungen vor. Ansible verfügt über eine Vielzahl an verschiedenen Modulen.

Mehr Informationen zu den seit Version 3.0 verfügbaren *Collections* finden Sie ab S. 264.

Mit dem Modul `package` lassen sich Pakete mittels des auf dem Zielhost vorhandenen Paketmanagers verwalten. Dazu gehören das Installieren, Aktualisieren und Entfernen von Paketen. Über Argumente können Sie die Version oder den zu verwendenden Paketmanager bestimmen. Wird keine Angabe gemacht, so ermittelt Ansible automatisch den Paketmanager und installiert die aktuellste Version des ausgewählten Pakets.

```
$ ansible ubuntu1 -i inventory -k -m ansible.builtin.package -a
  ↳ "name=nano"

$ ansible ubuntu1 -i inventory -k -m ansible.builtin.package -a
  ↳ "state=latest name=nano use=apt"
```



**Tipp** *Auflistung der verfügbaren Module in Ansible*

Mit dem Befehl `ansible-doc --list` bzw. `-l` können Sie sich eine Liste der verfügbaren Module ausgeben lassen. Mit `ansible-doc <modulname>` lassen Sie sich detaillierte Informationen zu dem bestimmten Modul anzeigen.

## Ansible-Module 2/2

### Das Modul setup ohne Argumente

```
$ ansible all -i inventory -m ansible.builtin.setup
```

### Das Modul user mit Argumenten

```
$ ansible suse1 -i prod -m ansible.builtin.user \
-a "name=tux state=present"
```

### Ausgabe einer Liste aller Module

```
$ ansible-doc --list
```

### Details und Argumente eines bestimmten Moduls ausgeben lassen

```
$ ansible-doc <modulename>
```

Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de  
 Mit dem Modul `setup` lassen sich detaillierte Informationen über das Zielsystem abfragen. Bei diesen Informationen handelt es sich um die sogenannten *Ansible Facts*. Diese Facts werden bei jedem Aufruf eines Playbooks standardmäßig abgerufen und können so über Variablen im Playbook verwendet werden. Neben dem automatischen Aufruf mit dem Befehl `ansible-playbook` kann das Modul auch in einem Ad-Hoc Befehl verwendet werden. Ohne Optionen aufgerufen listet es alle verfügbaren Facts über das Ziel- system in der Konsole auf:

```
$ ansible ubuntul -i /home/tux/ansible/inventory -m
  ↪ ansible.builtin.setup

ubuntul | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "172.20.1.121"
        ],
        [...]
    },
    "ansible_python_version": "2.7.15rc1",
    "ansible_real_group_id": 0,
    "ansible_real_user_id": 0,
    "ansible_selinux": {
        "status": "Missing selinux Python library"
    },
    "ansible_selinux_python_present": false,
```

```

"ansible_service_mgr": "systemd",
"ansible_ssh_host_key_ecdsa_public":
    ↪ "AAAAE2VjZHNh[...]vChqIhS0CCC8=",
"ansible_ssh_host_key_ed25519_public":
    ↪ "AAAAC3NzaC1lZDI1NTE5[...]tG+eQSZacZZXjoyzbSiN",
"ansible_ssh_host_key_rsa_public":
    ↪ "AAAAB3NzaC1y[...]qW9cIEXiZ/T",
"ansible_swapfree_mb": 2047,
"ansible_swaptotal_mb": 2047,
"ansible_system": "Linux",
"ansible_system_vendor": "LENOVO",
"ansible_uptime_seconds": 169889,
"ansible_user_dir": "/root",
"ansible_user_gecos": "root",
"ansible_user_gid": 0,
"ansible_user_id": "root",
"ansible_user_shell": "/bin/bash",
"ansible_user_uid": 0,
"ansible_userspace_architecture": "x86_64",
"ansible_userspace_bits": "64",
"ansible_virtualization_role": "guest",
"ansible_virtualization_type": "docker",
"gather_subset": [
    "all"
],
"module_setup": true
},
"changed": false
}

```

Mit der Option `filter` kann die Ausgabe auf bestimmte *Facts* begrenzt werden. So gibt der Filter `ansible_all_ipv4_addresses` die Facts über IPv4-Adressen aus:

```

$ ansible ubuntu -i /home/tux/ansible/inventory -m
    ↪ ansible.builtin.setup -a "filter=ansible_all_ipv4_addresses"
ubuntul | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "172.20.1.121"
        ]
    },
    "changed": false
}
ubuntu2 | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "172.20.1.122"
        ]
    },
    "changed": false
}

```

Der Filter `ansible_kernel` liefert Informationen über die Kernelversion:

## 14 Ansible Ad-Hoc Befehle

```
$ ansible centos -i /home/tux/ansible/inventory -m
  ↪ ansible.builtin.setup -a "filter=ansible_kernel"
centos2 | SUCCESS => {
    "ansible_facts": {
        "ansible_kernel": "5.0.7-200.fc29.x86_64"
    },
    "changed": false
}
centos1 | SUCCESS => {
    "ansible_facts": {
        "ansible_kernel": "5.0.7-200.fc29.x86_64"
    },
    "changed": false
}
```

Bei jedem Ausführen des Befehls `ansible-playbook` wird das `setup`-Modul mit ausgeführt, sodass die verfügbaren Facts über die Zielsysteme im Playbook selbst als Variablen zur Verfügung stehen.

### 14.5.1 Aufgabenteil



## Aufgabenteil – Module

- ① Verteilen Sie Ihren SSH-Key mit dem Modul `ansible.posix.authorized_key` auf alle Zielsysteme des Inventory.
- ② Legen Sie mit dem Modul `ansible.builtin.user` auf den Systemen der Gruppe `ubuntu` und auf `suse1` den Benutzer `tux` an, der die Shell `bash` benutzen soll.
- ③ Lassen Sie sich mit dem Modul `ansible.builtin.setup` die facts über die Netzwerkschnittstelle `eth0` (`ansible_eth0`) für die Gruppe `centos` ausgeben.

### Hinweis

Die notwendigen Argumente für die verwendeten Module lassen sich mit `ansible-doc` nachschlagen.

1. Verteilen Sie Ihren SSH-Key mit dem Modul `ansible.posix.authorized_key` auf alle Zielsysteme des Inventory.
2. Legen Sie mit dem Modul `ansible.builtin.user` auf den Systemen der Gruppe `ubuntu` und auf `suse1` den Benutzer `tux` an, der die Shell `bash` benutzen soll.
3. Lassen Sie sich mit dem Modul `ansible.builtin.setup` die facts über die Netzwerkschnittstelle `eth0` (`ansible_eth0`) für die Gruppe `centos` ausgeben.



### Tipp `ansible-doc`

Die notwendigen Argumente für die verwendeten Module lassen sich mit `ansible-doc` nachschlagen.

## Musterlösung

**1. Verteilen Sie Ihren SSH-Key mit dem Modul  
ansible.posix.authorized\_key auf alle Zielsysteme des Inventory:**

Der Schlüssel muss aus der Schlüsseldatei in `~/.ssh/` ausgelesen werden, damit er mit dem Modul `ansible.posix.authorized_key` auf das Zielsystem kopiert werden kann. Die Ausgabe von z. B. `cat` kann direkt in einer Subshell als Parameter an das Modul übergeben werden:

```
$ ansible all -k -i /home/tux/ansible/inventory -m
  ↪ ansible.posix.authorized_key -a "user=root key='$(cat
  ↪ ~/.ssh/id_rsa.pub)'"

SSH password:
ubuntul | CHANGED => {
    "changed": true,
    "comment": null,
    "exclusive": false,
    "follow": false,
    "key": "ssh-rsa AAAAB3Nza[...]NHDC0J4x tux@ansible",
    "key_options": null,
    "keyfile": "/home/tux/.ssh/authorized_keys",
    "manage_dir": true,
    "path": null,
    "state": "present",
    "unique": false,
    "user": "root",
    "validate_certs": true
}
centos2 | CHANGED => {
    "changed": true,
    "comment": null,
    "exclusive": false,
    "follow": false,
    "key": "ssh-rsa AAAAB3Nza[...]NHDC0J4x tux@ansible",
    "key_options": null,
    "keyfile": "/home/tux/.ssh/authorized_keys",
    "manage_dir": true,
    "path": null,
    "state": "present",
    "unique": false,
    "user": "root",
    "validate_certs": true
}
centos1 | CHANGED => {
    "changed": true,
    "comment": null,
    "exclusive": false,
    "follow": false,
    "key": "ssh-rsa AAAAB3Nza[...]NHDC0J4x tux@ansible",
    "key_options": null,
    "keyfile": "/home/tux/.ssh/authorized_keys",
    "manage_dir": true,
```

```

    "path": null,
    "state": "present",
    "unique": false,
    "user": "root",
    "validate_certs": true
}
suse1 | CHANGED => {
    "changed": true,
    "comment": null,
    "exclusive": false,
    "follow": false,
    "key": "ssh-rsa AAAAB3Nza[...]NHDC0J4x tux@ansible",
    "key_options": null,
    "keyfile": "/home/tux/.ssh/authorized_keys",
    "manage_dir": true,
    "path": null,
    "state": "present",
    "unique": false,
    "user": "root",
    "validate_certs": true
}
suse2 | CHANGED => {
    "changed": true,
    "comment": null,
    "exclusive": false,
    "follow": false,
    "key": "ssh-rsa AAAAB3Nza[...]NHDC0J4x tux@ansible",
    "key_options": null,
    "keyfile": "/home/tux/.ssh/authorized_keys",
    "manage_dir": true,
    "path": null,
    "state": "present",
    "unique": false,
    "user": "root",
    "validate_certs": true
}
ubuntu2 | CHANGED => {
    "changed": true,
    "comment": null,
    "exclusive": false,
    "follow": false,
    "key": "ssh-rsa AAAAB3Nza[...]NHDC0J4x tux@ansible",
    "key_options": null,
    "keyfile": "/home/tux/.ssh/authorized_keys",
    "manage_dir": true,
    "path": null,
    "state": "present",
    "unique": false,
    "user": "root",
    "validate_certs": true
}

```

Der Key wird in diesem Fall direkt in einer Subshell mit `cat` ausgelesen und als Argument an das Modul `ansible.posix.authorized_key` übergeben, welches

den Key dann auf den Zielsystemen einträgt.

2. Legen Sie mit dem Modul `ansible.builtin.user` auf den Systemen der Gruppe `ubuntu` und auf `suse1` den Benutzer `tux` an, der die Shell `bash` benutzen soll:

```
$ ansible ubuntu,suse1 -i /home/tux/ansible/inventory -m
  ↪ ansible.builtin.user -a "name=tux shell=/bin/bash
  ↪ state=present"

ubuntu2 | SUCCESS => {
    "append": false,
    "changed": false,
    "comment": "",
    "group": 1000,
    "home": "/home/tux",
    "move_home": false,
    "name": "tux",
    "shell": "/bin/bash",
    "state": "present",
    "uid": 1000
}
ubuntul | SUCCESS => {
    "append": false,
    "changed": false,
    "comment": "",
    "group": 1001,
    "home": "/home/tux",
    "move_home": false,
    "name": "tux",
    "shell": "/bin/bash",
    "state": "present",
    "uid": 1001
}
suse1 | SUCCESS => {
    "append": false,
    "changed": false,
    "comment": "",
    "group": 100,
    "home": "/home/tux",
    "move_home": false,
    "name": "tux",
    "shell": "/bin/bash",
    "state": "present",
    "uid": 1003
}
```

Sie können sich als `root` auf einem der Systeme einloggen, um zu überprüfen, ob der Benutzer angelegt wurde:

```
# ssh root@ubuntul
# su - tux
```

3. Lassen Sie sich mit dem Modul `ansible.builtin.setup` die `facts` über die Netzwerkschnittstelle `eth0 (ansible_eth0)` für die Gruppe `centos` ausgeben:

```
$ ansible centos -i /home/tux/ansible/inventory -m
  ↪ ansible.builtin.setup -a "filter=ansible_eth0"

centos1 | SUCCESS => {
    "ansible_facts": {
        "ansible_eth0": {
            "active": true,
            "device": "eth0",
            "ipv4": {
                "address": "172.20.1.101",
                "broadcast": "global",
                "netmask": "255.255.255.0",
                "network": "172.20.1.0"
            },
            "ipv6": [
                {
                    "address": "fe80::42:acff:fe14:165",
                    "prefix": "64",
                    "scope": "link"
                }
            ],
            "macaddress": "02:42:ac:14:01:65",
            [...]
        }
    },
    "changed": false
}
centos2 | SUCCESS => {
    "ansible_facts": {
        "ansible_eth0": {
            "active": true,
            "device": "eth0",
            "ipv4": {
                "address": "172.20.1.102",
                "broadcast": "global",
                "netmask": "255.255.255.0",
                "network": "172.20.1.0"
            },
            "ipv6": [
                {
                    "address": "fe80::42:acff:fe14:166",
                    "prefix": "64",
                    "scope": "link"
                }
            ],
            "macaddress": "02:42:ac:14:01:66",
            [...]
        }
    },
    "changed": false
}
```

## 14.6 Ansible Collections



# Ansible Collections

Seit Ansible Version 2.10:

- Unterscheidung zwischen:
  - `ansible-core`
  - `ansible`
- `ansible` beinhaltet `ansible-core` plus viele Collections/Module
- Collections enthalten (u. a.) Module
  - werden mit `ansible-galaxy` verwaltet
  - sind versioniert

Mit dem Versionssprung von Ansible Version 2.9 auf Version 2.10 hat Red Hat die Struktur der Installation sowie die Handhabung der Module überarbeitet. Der Hintergrund dieser Entscheidung liegt in der stetig wachsenden Popularität von Ansible, die dazu geführt hat, dass die Anzahl der verfügbaren Module rasant gewachsen ist (und wächst). In den Versionen bis Ansible 2.9 wurden Aktualisierungen von Modulen mit dem nächsten Ansible Release ausgeliefert. Dieses Vorgehen hatte die Nachteile, dass Modul-Updates entweder erst mit dem nächsten Release ausgeliefert wurden oder im schlimmsten Fall selbst einen neuen Release erzwungen haben, weil das Update nicht warten konnte. Daraus wurde mit Ansible Version 2.10 der größte Anteil der zuvor in der Standardinstallation enthaltenen Module in sogenannte *Collections* ausgegliedert, die zusätzlich installiert werden können. Dies hat den Vorteil, dass Module nun unabhängig von Ansible gepflegt werden können und somit nicht mehr vom Releasezyklus von Ansible abhängig sind.

Mit der Ausgliederung der Module in Collections haben sich zudem auch die Installationsmöglichkeiten von Ansible und damit einhergehend die Art der Versionierung verändert. Ansible steht jetzt in den Paketen `ansible-core` und `ansible` zur Installation bereit. `ansible-core` enthält die Binaries sowie einen Grundstock an Modulen, wohingegen mit `ansible` das Paket `ansible-core` plus eine große Menge von Collections und damit Modulen installiert wird. Grundsätzlich können Collections mit dem Werkzeug `ansible-galaxy` verwaltet werden.



# Ansible Versionierungsschema

- Ansible
  - ansible-core als Abhängigkeit
  - nutzt semantisches Versionierungsschema:  
Ansible <MAJOR>.<MINOR>.<PATCH>, z. B.  
Ansible 5.2.1
  - Ansible 4 ist abhängig von ansible-core 2.11
  - Ansible 5 ist abhängig von ansible-core 2.12
  - ...
- ansible-core (war bis 2.10 ansible-base)
  - nutzt *kein* semantisches Versionierungsschema, z. B.
    - 2.11
    - 2.12
    - ...

Mit der Einführung von Collections und der Aufteilung der Installationspakete in ansible-core (bis Version 2.10 ansible-base genannt) und Ansible führte Red Hat auch ein neues Versionierungsschema ein. Seitdem werden immer jeweils zwei Installationskandidaten veröffentlicht: Zum einen ansible-core, welches die Kernkomponenten beinhaltet und einem fortlaufenden Versionierungsschema folgt, zum anderen Ansible, welches ansible-core als Abhängigkeit hat und einem semantischen Versionierungsschema folgt um flexibler innerhalb derselben Version Anpassungen vornehmen zu können. Die Versionsnummern der semantischen Versionierung folgen dabei dem Muster <MAJOR>.<MINOR>.<PATCH>. Bei einer Versionsnummer von z. B. 5.1.2 würde dies bedeuten, dass es sich um den zweiten Patch im ersten Minor-Release des fünften Major-Release handelt. Zudem gilt dabei, dass die Major-Releases nicht kompatibel zu vorherigen Versionen sind, die Minor-Releases sowie die Patches allerdings schon (für diese Major-Version). Der Major-Release der Ansibleversionen ist dabei abhängig von der gegenwärtigen Version von ansible-core. So ist Ansible 3.X.X abhängig von ansible-core 2.10, Ansible 4.X.X von ansible-core 2.11, 5.X.X von 2.12, usw.. Seit ansible-core Version 2.12 wird Python in der Version >=3.8 vorausgesetzt.

## Collections verwalten

- ansible-galaxy: Werkzeug zur Verwaltung von Ansible-Quellen
- u. a. Verwaltung von Collections

### Installation einer Collection samt Modulen

```
$ ansible-galaxy collection install community.docker
```

### Installation einer Collection in bestimmter Version

```
$ ansible-galaxy collection install amazon.aws==1.4.2
```

### Installierte Collections anzeigen lassen

```
$ ansible-galaxy collection list
```

Mit dem Programm `ansible-galaxy`, das standardmäßig in `ansible-core` enthalten ist, steht Ihnen ein Werkzeug zur Verwaltung von Collections zur Verfügung. Über den Parameter `collection` nutzen Sie `ansible-galaxy` um Operationen mit und an Collections durchzuführen. Die Option `--help (-h)` liefert eine Auflistung aller möglichen Operationen:

```
$ ansible-galaxy collection --help
```

Die wichtigsten Operationen die Handhabung von Modulen betreffend sind `install` und `list`. Mit `install` installieren Sie eine Collection, mit `list` erhalten Sie eine Liste der bereits installierten Collections. Da Sie mit dem `ansible-galaxy`-Befehl nicht nach Collections suchen können, ist es notwendig den Namen der Collection bereits vor der Installation zu kennen. Hierfür suchen Sie nach Bedarf in der offiziellen Ansible Dokumentation nach den gewünschten Modulen:

<https://docs.ansible.com/ansible/latest/collections/index.html>

Auf der Dokumentationsseite des gewünschten Moduls finden Sie einen Hinweis darüber, in welcher Collection sich das Modul befindet und eine Beschreibung, wie Sie das Modul mit Hilfe von `ansible-galaxy` installieren:

## community.postgresql.postgresql\_copy – Copy data between a file/program and a PostgreSQL table

**Note**

This plugin is part of the [community.postgresql collection](#) (version 1.6.0).

You might already have this collection installed if you are using the `ansible` package. It is not included in `ansible-core`. To check whether it is installed, run `ansible-galaxy collection list`.

To install it, use: `ansible-galaxy collection install community.postgresql`.

To use it in a playbook, specify: `community.postgresql.postgresql_copy`.

### Installation von Collections

Um das aus dem obigen Beispiel gezeigte Community-Modul `community.postgresql.postgresql_copy` zu installieren, führen Sie den folgenden Befehl aus:

```
$ ansible-galaxy collection install community.postgresql
```

Das gewünschte Modul wird als Teil der Collection mit auf Ihrem System installiert.

### Installation von Collections in bestimmten Versionen

Wie bereits angedeutet und auf dem Screenshot der Ansible-Dokumentation zu erkennen, werden Collections versioniert. Dies bedeutet, dass Sie bei Bedarf bestimmte Versionen der Collections installieren können um bspw. Konflikte in Ihrem Ansible-Code zu vermeiden. Eine präzise Version der Collection geben Sie bei der Installation mit dem folgenden Befehl an:

```
$ ansible-galaxy collection install amazon.aws==1.4.2-dev9 --force
```

Dem eigentlichen Installationsbefehl wird mittels `==<MAJOR>.<MINOR>.<PATCH>` nach dem semantischen Versionierungsschema die genaue Version mit angegeben. Die Option `--force` ist dann notwendig, wenn die Collection bereits in einer anderen Version installiert ist, da `ansible-galaxy` diese sonst nicht überschreibt.

## Installation von Collections aus Git-Repositories

Zusätzlich zu der Option, Collections aus der Ansible Galaxy Datenbank zu installieren, können Sie Collections mit der folgenden Syntax auch direkt aus Git-Repositories installieren:

```
$ ansible-galaxy collection install
  ↪ git+https://github.com/tuxpinguin/repo_name.git,stable
```

Hierfür übergeben Sie dem `ansible-galaxy`-Installationsbefehl nicht den Namen der gewünschten Collection, sondern die URL des Repository im Schema `git+<Repository-URL>, <Name_des_Branches>`.

## Anzeigen von installierten Collections

Um eine Übersicht darüber zu erhalten welche Collections Sie bereits in Ihrer Ansible-Umgebung installiert haben führen Sie den folgenden Befehl aus:

```
$ ansible-galaxy collection list

# /home/tux/.ansible/collections/ansible_collections
Collection          Version
-----
amazon.aws          3.1.1
community.docker    2.1.1
community.postgresql 2.1.0
```

Über den Parameter `list` lassen Sie eine Liste ausgeben, welche Ihnen zum einen das Installationsverzeichnis (im Beispiel `/root/.ansible/collections/ansible_collections`) und zum anderen die installierten Collections samt deren Version ausgibt. Standardmäßig werden manuell installierte Collections im Home-Verzeichnis des Users unter `/home/<user>/ ansible/collections/ansible_collections` abgelegt.

## Optionen für `ansible-galaxy collection`

Für den Befehl `ansible-galaxy collection` sind folgende Optionen verfügbar:

OPTION	BEDEUTUNG
download	Lädt die Collection als Tarball zur manuellen Installation herunter.
init	Erzeugt die leere Struktur einer neuen Collection.
build	Erzeugt ein Collection-Artefakt* zur Publikation in der Galaxy.
publish	Publiziert ein Collection-Artefakt* in der Galaxy.
install	Installiert Collections aus Dateien oder von URLs.
list	Gibt eine Liste der installierten Collections aus.
verify	Gleicht Checksummen von lokalen Collections mit der Galaxy ab.

\* Als Artefakt wird hier ein `.tar.gz`-Archiv verstanden.



**Tipp** Pfad der Ansible Collections bei der Installation des Pakets `ansible`  
Sofern die Collections in der Paketinstallation des Pakets `ansible` installiert wurden,  
lautet der Standardinstallationspfad  
`/usr/local/lib/python3.x/site-packages/ansible_collections.`

## Collection Namespaces

- Benennung von Collections folgt einem bestimmtem Aufbau:
  - <namespace>.<collection>
  - Beispiel `ansible.posix`
- für Module:
  - <namespace>.<collection>.<module>
  - Beispiel `ansible.posix.mount`

Module aus bestimmter Collection ausgeben lassen:

```
$ ansible-doc --list ansible.posix
```

Informationen über ein Modul einer Collection erhalten:

```
$ ansible-doc ansible.posix.mount
```

Ansible Collections werden einem Namespace zugeordnet, über den sie referenzierbar sind. Jede Collection wird dabei einem einzigen Namespace zugeordnet, wohingegen ein Namespace eine Vielzahl unterschiedlicher Collections beinhalten kann. Die Referenz verläuft dabei immer nach dem Muster `<namespace>.<collection>`. Im folgenden Beispiel bezeichnet `ansible` den Namespace und `posix` die Collection:

```
ansible.posix
```

Unter der Angabe des Namespaces gibt der Befehl `ansible-doc --list` die Module einer bestimmten Collection aus:

```
$ ansible-doc --list ansible.posix
ansible.posix.acl           Set and retrieve file ACL information
ansible.posix.at              Schedule the execution of a command[...]
ansible.posix.authorized_key Adds or removes an SSH authorized key
ansible.posix.firewalld       Manage arbitrary ports/services [...]
[...]
```

Die hier gelisteten Module sind Teil der Collection `posix`, die sich wiederum im Namespace `ansible` befindet. Analog dazu gehört das Modul aus dem folgenden Beispiel ebenfalls zum Namespace `ansible`, jedoch in der Collection `builtin`:

```
$ ansible susel -i inventory -m ansible.builtin.ping
```

Informationen über ein bestimmtes Modul erhalten Sie mit dem Befehl `ansible-doc` unter der Angabe von Namespace, Collection sowie des Namens des Moduls:

```
$ ansible-doc ansible.posix.mount
```

# Migration zu neuen Ansible-Versionen

Upgrade auf eine neue Ansible-Version:

- mittels distributionseigenem Paketmanager oder
- per PIP
- ab ansible-core 2.12 wird python $\geq$ 3.8 benötigt
- Code und Module in neuer Version testen
- Pro Release Changelogs lesen

**Fehlermeldung des ansible.builtin.package Moduls**

"Could not find a module for zypper."

**Fehlermeldung aufgrund eines Modul-Umzugs:**

"The module ansible.builtin.zypper was redirected to \\ community.general.zypper, which could not be loaded."

Bei der Migration von früheren Ansible-Versionen zur aktuellen Version mit ansible-core 2.12 sind einige Punkte zu beachten.

## Upgrade auf neue Ansible-Version

Das Upgrade auf eine neue Ansible-Version mittels des distributionseigenen Paketmanagers beinhaltet lediglich eine Deinstallation der vorhandenen Ansible-Version, gefolgt von der Installation der gewünschten Version.

Wurde die Installation mit pip durchgeführt, so kann die Upgrade-Option (-U) von pip genutzt werden:

### Upgrade auf die neuste Version:

```
# /usr/bin/python3 -m pip install -U ansible
```

### Upgrade auf eine bestimmte Version:

```
# /usr/bin/python3 -m pip install -U ansible-core==2.12.2
```

## Fehlende Modulabhängigkeiten

Durch die Einführung der Collections kann es dazu kommen, dass vorher über die Standardinstallation vorhandene Module nun in Abhängigkeitsprobleme geraten, weil sie in Collections ausgelagert wurden. Ein solcher Fall ist bspw. das package-Modul, das auf den Zielsystemen einfache Paketoperationen (install, upgrade, remove) durchführt. Bei dem Versuch, auf einem SUSE-basierten System ein Paket zu installieren, erscheint nun die folgende Nachricht:

```
"Could not find a module for zypper."
```

Da das Modul ansible.builtin.package auf dem Zielsystem auf den dort bereits vorhandenen Paketmanager zurückgreift, benutzt es hierfür das für diesen Paketmanager zuständige Ansible-Modul, in diesem Falle das Modul zypper. Wird das Modul zypper einzeln aufgerufen, z. B. in einem Ad-Hoc Befehl, so gibt Ansible einen Hinweis aus:

```
"The module ansible.builtin.zypper was redirected to
  ↵ community.general.zypper, which could not be loaded."
```

In diesem Fall ist es notwendig das Modul zypper beziehungsweise dessen Collection zu installieren:

```
# ansible-galaxy collection install community.general
```

Im Anschluss kann auch das ansible.builtin.package-Modul wieder auf das community.general.zypper-Modul zugreifen und somit Paketoperationen an SUSE-Systemen durchführen.

## Changelogs lesen

Es ist ratsam vor Upgrades die von Ansible sowie von der Community bereitgestellten Changelogs zu lesen um möglichen Fehlern durch Änderungen zuvorzukommen. In den Changelogs wird beispielsweise erwähnt, dass ansible-core ab Version 2.12 Python 3.8 voraussetzt. Zudem werden hier Änderungen in Collections dokumentiert, die ebenso Auswirkungen auf das Verhalten des eigenen Codes haben können.

Die Changelogs sind unter der folgenden Adresse einsehbar:

```
https://docs.ansible.com/ansible-devel/reference\_appendices/release\_and\_maintenance.html
```



#### 14.6.1 Aufgabenteil



## Aufgabenteil – Collections 1

- ① Installieren Sie die Collections `ansible.utils` und `ansible.posix`.
- ② Lassen Sie sich eine Liste der installierten Collections ausgeben.
- ③ Lassen Sie sich eine Liste aller Module aus der Collection `ansible.utils` ausgeben.
- ④ Installieren sie das Paket `ansible`.
- ⑤ Lassen Sie sich erneut eine Liste der installierten Collections ausgeben.

1. Installieren Sie die Collections `ansible.utils` und `ansible.posix`.
2. Lassen Sie sich eine Liste der installierten Collections ausgeben.
3. Lassen Sie sich eine Liste aller Module aus der Collection `ansible.utils` ausgeben.
4. Installieren sie das Paket `ansible`.
5. Lassen Sie sich erneut eine Liste der installierten Collections ausgeben.

## Musterlösung

### 1. Installieren Sie die Collections `ansible.utils` und `ansible.posix`:

Führen Sie den Befehl `ansible-galaxy collection install` jeweils einmal für jede der beiden Collections aus:

```
$ ansible-galaxy collection install ansible.utils
Starting galaxy collection install process
Process install dependency map
Starting collection install process
Downloading https://galaxy.ansible.com/download/
    ↳ ansible-utils-2.5.0.tar.gz to
    ↳ /home/tux/.ansible/tmp/ansible-local-486318vlmmkdm/
    ↳ tmpqlc38swk/ansible-utils-2.5.0-b28ezjrh
Installing 'ansible.utils:2.5.0' to
    ↳ '/home/tux/.ansible/collections/ansible_collections/ansible/utils'
ansible.utils:2.5.0 was installed successfully

$ ansible-galaxy collection install ansible.posix
Starting galaxy collection install process
Process install dependency map
Starting collection install process
Downloading https://galaxy.ansible.com/download/
    ↳ ansible-posix-1.3.0.tar.gz to
    ↳ /home/tux/.ansible/tmp/ansible-local-48636vog0obw7/
    ↳ tmpjlpa8pgw/ansible-posix-1.3.0-qzokdwx8
Installing 'ansible.posix:1.3.0' to
    ↳ '/home/tux/.ansible/collections/ansible_collections/ansible posix'
ansible.posix:1.3.0 was installed successfully
```

### 2. Lassen Sie sich eine Liste der installierten Collections ausgeben:

Geben Sie die Liste der installierten Collections mit der Option `list` aus:

```
$ ansible-galaxy collection list

# /home/tux/.ansible/collections/ansible_collections
Collection      Version
----- -----
ansible.posix  1.3.0
ansible.utils  2.5.0
```

### 3. Lassen Sie sich eine Liste aller Module aus der Collection `ansible.utils` ausgeben:

Führen Sie den `ansible-doc` Befehl mit der Option `--list` unter Angabe von Namespace und Collection aus:

```
$ ansible-doc --list ansible.utils
ansible.utils.cli_parse  Parse cli output or text using a
    ↳ variety of parsers
ansible.utils факт_diff  Find the difference between currently
    ↳ set facts
ansible.utils.update_fact Update currently set facts
ansible.utils.validate    Validate data with provided criteria
```

#### 4. Installieren sie das Paket **ansible**:

Installieren Sie das Paket `ansible` mit dem Python Paketmanager `pip`:

```
# /usr/bin/python3 -m pip install ansible
```

#### 5. Lassen Sie sich erneut eine Liste der installierten Collections ausgeben:

Führen Sie erneut die Option `list` aus:

```
$ ansible-galaxy collection list
```

```
# /home/tux/.ansible/collections/ansible_collections
Collection      Version
-----
ansible.posix  1.3.0
ansible.utils  2.5.0
```

```
# /usr/local/lib/python3.8/site-packages/ansible_collections
Collection      Version
-----
amazon.aws       2.1.0
ansible.netcommon 2.5.0
ansible.posix     1.3.0
ansible.utils     2.4.3
ansible.windows   1.9.0
arista.eos        3.1.0
awx.awx           19.4.0
azure.azcollection 1.11.0
check_point.mgmt  2.2.2
chocolatey.chocolatey 1.1.0
cisco.aci          2.1.0
cisco.asa          2.1.0
[...]
```

Sie bekommen in der Ausgabe nun zwei Listen angezeigt – zum einen die der manuell installierten Collections im Home-Verzeichnis, und zum anderen die Liste der Collections, welche mit dem `ansible`-Paket installiert wurden. Beachten Sie, dass sich die Versionen der Collections zwischen den beiden Installationsarten unterscheiden können. So liegt die Collection `ansible.utils` in diesem Falle in der manuelle Installation in der Version `2.5.0`, in der Installation aus dem `ansible`-Paket jedoch in der Version `2.4.3` vor.

## 15 Ansible Playbooks



# Ansible Playbooks

B1 Systems GmbH

Persönliches Exemplar von Peter.Wohlfarth@justiz-thueringen.de

## 15.1 Zielsetzung



# Zielsetzung

Dieses Kapitel liefert einen Einstieg in Playbooks:

- Aufbau eines Playbook
- Nutzung von Variablen
  - Welche Variablentypen gibt es?
  - Wo können Variablen gesetzt werden?

## 15.2 Playbooks



# Was ist ein Playbook?

- kann eine ganze Umgebung beschreiben
- kann mehrere *Plays* umfassen
- Plays beinhalten:
  - Zieldefinition
  - Variablen
  - Handlers
  - Optionen
  - Tasks

Bei *Playbooks* handelt es sich in Ansible um das Format, in dem komplexe Aufgaben strukturiert verfasst werden können. Im Gegensatz zu Ad-Hoc Befehlen kann im Playbook auch bequem mit Variablen und komplexen Abhängigkeiten gearbeitet werden. Playbooks enthalten dabei eine Liste von Plays, in denen wiederum Hosts, Variablen und Tasks deklariert werden können. Durch die Möglichkeit, Plays und Tasks modular einzubinden, können Playbooks übersichtlich gestaltet werden, ohne auf die notwendige Komplexität bei der Gestaltung der Ziellandschaft verzichten zu müssen.



# Aufbau eines Playbook

- YAML-Syntax
- enthalten *Listen* und *Dictionaries*
- Liste von Plays
- Deklaration von:
  - Hosts
  - Variablen (optional)
  - einzelnen Tasks

Der grobe Aufbau eines Playbook:

```
---
- name: The name of the play
  hosts: all
  vars:
    variable1: tux
  tasks:
    - name: first task
      ansible.builtin.user:
        name: "{{ variable1 }}"
        state: present
      register: user_result
    - name: second task
      ansible.builtin.package:
        name: nano
        state: latest
```

Playbooks werden in YAML-Syntax (*YAML Ain't Markup Language*) verfasst und enthalten dementsprechend YAML-*Listen* und *Dictionaries*. In Playbooks werden Plays erstellt, die Parameter wie Zielsysteme (`hosts:`) oder Variablen (`vars:`) enthalten. In diesen Plays wird mit `tasks:` ein *Dictionary* eröffnet, das eine Liste mit Tasks enthält. In den

## 15 Ansible Playbooks

Tasks werden Module (`ansible.builtin.user`, `ansible.builtin.package`) verwendet, die wiederum über eigene Parameter (`name`, `state`) verfügen. Innerhalb von Playbooks gibt es viele verschiedene Möglichkeiten, die einzelnen Plays zu gestalten. Das hier gezeigte Playbook enthält ein Play mit zwei Tasks. Es ist möglich, mit einer Liste von Plays eine ganze Ziellandschaft mit einem einzigen Playbook zu beschreiben.



# YAML-Syntax in Playbooks

## YAML-Syntax

```
---
- name: My first playbook
  hosts: all
  tasks:
    - name: ensure apache is installed
      ansible.builtin.package:
        name: httpd
        state: latest
```

## Komprimierte Syntax

```
- name: ensure apache is installed
  ansible.builtin.package: name=httpd state=latest
```

YAML-Dateien beginnen immer mit --- und enthalten dann sogenannte *Dictionaries*, in denen zusammengehörige Abschnitte zusammengefasst werden. In den *Dictionaries* werden Werte nach dem *Schlüssel: Wert* (Key: Value) Prinzip festgelegt. Achten Sie beim Verfassen von YAML-Dateien darauf, Leerzeichen statt Tabstops zu benutzen und darauf, dass YAML zwischen Groß- und Kleinschreibung unterscheidet.

Beispiel für ein Dictionary:

```
- name: ensure apache is installed
  ansible.builtin.package:
    name: 'httpd'
    state: latest
```

Grundsätzlich ist es egal, in welcher Reihenfolge die einzelnen Einträge in den jeweiligen *Dictionaries* stehen, da diese nach ihrer hierarchischen Anordnung ausgewertet werden. Somit wird der folgende Eintrag genauso ausgewertet wie der vorherige:

```
- ansible.builtin.package:
  state: latest
  name: 'httpd'
name: ensure apache is installed
```



### Hinweis Komprimierte Syntax als Alternative zu YAML

Neben dem Verfassen der Playbooks in der YAML-Syntax besteht auch die Möglichkeit, eine komprimierte Syntax – in Anlehnung an die Syntax der Ad-Hoc-Befehle – zu verwenden. Es ist allerdings generell empfohlen, die YAML-Syntax zu nutzen, da diese eine deutlich bessere Übersichtlichkeit gewährleistet, was gerade bei längeren Playbooks einen großen Vorteil bedeuten kann.



# Playbooks – Struktur

- Host-Deklaration
- Variablen (optional)
- Aufgabenbeschreibung (Tasks)
- Aufruf eines Playbooks mit ansible-playbook

## Playbooks – Struktur

```
---
- name: Unser erstes Playbook
  hosts: all
  vars:
    variable: value
  tasks:
    - name: Name der Aufgabe
      ansible.builtin.package:
        name: nano
        state: latest
```

Ein Playbook enthält immer mindestens die Angabe der Zielsysteme (`hosts:`) und ein auszuführendes Modul (unter `tasks:`). Die angegebenen Hosts müssen zuvor im Inventory eingetragen werden. Die Inventory-Datei wird beim Ausführen von `ansible-playbook` mit der Option `-i` übergeben. Der Dictionary-Eintrag `tasks:` enthält Module und ggf. Parameter.

Der Aufruf bzw. die Ausführung eines Playbooks erfolgt mit dem Befehl `ansible-playbook`, z.B.:

```
# ansible-playbook -i inventory playbook.yml
```

Der Aufbau eines Playbooks:

```
- name: Unser erstes Playbook
  hosts: all
  vars:
    variable: value
  tasks:
    - name: Name der Aufgabe
      ansible.builtin.package:
        name: nano
        state: latest
```

In diesem Fall werden die im Playbook enthaltenen `tasks` durch die Angabe von `all` bei `hosts` für alle im Inventory eingetragenen Systeme ausgeführt. Unter `vars:` können

## 15 Ansible Playbooks

Variablen deklariert werden, die dann innerhalb des Playbooks gültig sind. Hier wird die Variable `variable` mit dem Wert `value` deklariert. Schließlich wird in einem Task das Modul `ansible.builtin.package` ausgeführt, welches sicherstellt, dass die aktuellste Version des Programms `nano` auf den Zielsystemen vorliegt.



### Tipp Targeting-Optionen beim Ausführen von Playbooks

Beim Ausführen von Playbooks mit dem Befehl `ansible-playbook` lassen sich dieselben Targeting-Optionen anwenden wie für Ad-Hoc Befehle.

### 15.2.1 Aufgabenteil



## Aufgabenteil – Playbook

Schreiben Sie ein Playbook, in dem ein Task mit dem Modul `ansible.builtin.file` sicherstellt, dass das Verzeichnis `/home/tux/foo/` auf allen Zielsystemen vorhanden ist.

Schreiben Sie ein Playbook, in dem ein Task mit dem Modul `ansible.builtin.file` sicherstellt, dass das Verzeichnis `/home/tux/foo/` auf allen Zielsystemen vorhanden ist.

### Musterlösung

Schreiben Sie ein Playbook, in dem ein Task mit dem Modul `ansible.builtin.file` sicherstellt, dass das Verzeichnis `/home/tux/foo/` auf allen Zielsystemen vorhanden ist.:

Beispiel: Playbook `/root/code/file_foo.yml`:

```
---
- name: Playbook directory control
  hosts: all
  tasks:
    - name: ensure that the directory is present
      ansible.builtin.file:
        path: /home/tux/foo/
        state: directory
```

Das Playbook mit `ansible-playbook` ausführen:

```
# ansible-playbook -i inventory file_foo.yml

PLAY [Playbook directory control] **

TASK [Gathering Facts] **
ok: [ubuntu1]
ok: [ubuntu2]
ok: [suse2]
ok: [suse1]
ok: [centos1]
ok: [centos2]

TASK [ensure that the directory is present] **
changed: [ubuntu1]
changed: [centos2]
changed: [centos1]
changed: [suse2]
changed: [suse1]
changed: [ubuntu2]

PLAY RECAP **
centos1 : ok=2    changed=1    unreachable=0    failed=0
centos2 : ok=2    changed=1    unreachable=0    failed=0
suse1   : ok=2    changed=1    unreachable=0    failed=0
suse2   : ok=2    changed=1    unreachable=0    failed=0
ubuntu1 : ok=2    changed=1    unreachable=0    failed=0
ubuntu2 : ok=2    changed=1    unreachable=0    failed=0
```

### 15.3 Mehrere Tasks in einem Playbook



## Mehrere Tasks in einem Playbook

```

Mehrere Tasks in einem Playbook
-----
- name: Playbook with 2 tasks
  hosts: ubuntu
  tasks:
    - name: ensure that the directory is present
      ansible.builtin.file:
        path: /home/tux/foo/
        state: directory

    - name: make sure ssh key is placed
      ansible.posix.authorized_key:
        user: tux
        key: ssh-rsa AAAAB3Nz[...]J4x root@controller
        state: present

```

In Playbooks ist es möglich, mehrere Aufgaben nacheinander ausführen zu lassen. Dafür müssen Sie lediglich eine (YAML-)Liste mit den gewünschten tasks bereitstellen. Diese Liste wird dann von ansible-playbook von oben nach unten abgearbeitet. Wichtig zu beachten ist dabei, dass der gesamte Vorgang für das jeweilige Zielsystem abgebrochen wird, sobald ein Task nicht umsetzbar ist, da fehlerhafte Tasks nicht einfach übersprungen werden.

Playbook mit mehreren Tasks:

## 15 Ansible Playbooks

```
state: present
```

In diesem Playbook werden zwei Tasks ausgeführt. Der erste Task legt mit dem `file`-Modul ein Verzeichnis an, der zweite Task kopiert einen SSH-Key für den Benutzer `tux` auf das Zielsystem.

### 15.3.1 Aufgabenteil



## Aufgabenteil – Mehrere Tasks in einem Playbook

Schreiben Sie ein Playbook für alle Zielsysteme, das mit einem Task und dem Modul `ansible.builtin.user` sicherstellt, dass der Benutzer `tux` vorhanden ist.

Ein weiterer Task soll mit dem Modul `ansible.builtin.file` sicherstellen, dass das Verzeichnis `/home/tux/foo/` vorhanden ist und dem Benutzer `tux` gehört.

Erstellen Sie einen weiteren Task, in dem Sie mit dem Modul `ansible.builtin.copy` die Inventory-Datei in das angelegte Verzeichnis `/home/tux/foo/` kopieren lassen.

Schreiben Sie ein Playbook für alle Zielsysteme, das mit einem Task und dem Modul `ansible.builtin.user` sicherstellt, dass der Benutzer `tux` vorhanden ist.

Ein weiterer Task soll mit dem Modul `ansible.builtin.file` sicherstellen, dass das Verzeichnis `/home/tux/foo/` vorhanden ist und dem Benutzer `tux` gehört.

Erstellen Sie einen weiteren Task, in dem Sie mit dem Modul `ansible.builtin.copy` die Inventory-Datei in das angelegte Verzeichnis `/home/tux/foo/` kopieren lassen.

## Musterlösung

**Schreiben Sie ein Playbook für alle Zielsysteme, das mit einem Task und dem Modul `ansible.builtin.user` sicherstellt, dass der Benutzer `tux` vorhanden ist. Ein weiterer Task soll mit dem Modul `ansible.builtin.file` sicherstellen, dass das Verzeichnis `/home/tux/foo/` vorhanden ist und dem Benutzer `tux` gehört. Erstellen Sie einen weiteren Task, in dem Sie mit dem Modul `ansible.builtin.copy` die Inventory-Datei in das angelegte Verzeichnis `/home/tux/foo/` kopieren lassen:**

Beispiel für ein Playbook: `/root/code/file_copy_foo.yml`:

```
---
- name: Playbook a home for tux
  hosts: all
  tasks:
    - name: make sure tux is present
      ansible.builtin.user:
        name: tux
        state: present

    - name: make sure dir is present and belongs to tux
      ansible.builtin.file:
        path: /home/tux/foo/
        owner: tux
        state: directory

    - name: ensure inventory is copied to /home/tux/foo/
      ansible.builtin.copy:
        src: /root/code/inventory
        dest: /home/tux/foo/
```

Beim Ausführen des Playbooks arbeitet `ansible-playbook` die einzelnen Tasks für alle im Inventory eingetragenen Systeme ab:

```
# ansible-playbook -i inventory file_copy_foo.yml

PLAY [Playbook a home for tux] **

TASK [Gathering Facts] **
ok: [ubuntu1]
ok: [ubuntu2]
ok: [suse2]
ok: [suse1]
ok: [centos1]
ok: [centos2]

TASK [make sure tux is present] **
ok: [ubuntu1]
ok: [ubuntu2]
ok: [suse1]
ok: [suse2]
changed: [centos1]
changed: [centos2]
```

```
TASK [make sure dir is present and belongs to tux] **
ok: [ubuntul]
ok: [centos1]
ok: [centos2]
ok: [suse2]
ok: [suse1]
ok: [ubuntu2]

TASK [ensure inventory is copied to /home/tux/foo/] **
changed: [ubuntul]
changed: [centos1]
changed: [centos2]
changed: [suse2]
changed: [suse1]
changed: [ubuntu2]

PLAY RECAP **
centos1 : ok=4    changed=2    unreachable=0
          ↪ failed=0
centos2 : ok=4    changed=2    unreachable=0
          ↪ failed=0
suse1   : ok=4    changed=1    unreachable=0
          ↪ failed=0
suse2   : ok=4    changed=1    unreachable=0
          ↪ failed=0
ubuntul: ok=4    changed=1    unreachable=0
          ↪ failed=0
ubuntu2: ok=4    changed=1    unreachable=0
          ↪ failed=0
```

## 15.4 Variablen



# Variablen

Ansible unterstützt eine Reihe verschiedener Variablen, u. a.

- Variablen im Inventory
- `host_vars`
- `group_vars`
- `ansible_facts`
- Variablen im Playbook
  - registrierte Variablen
- Variablen auf der Kommandozeile

Ansible unterstützt die Deklaration von Variablen in verschiedenen Kontexten:

- Variablen können beim Aufruf von Ansible auf der Kommandozeile übergeben werden.
- Variablen können direkt im Inventory oder in Playbooks deklariert werden.
- Die sogenannten `ansible_facts` werden von Ansible zu Verfügung gestellt.

Diese verschiedenen Typen von Variablen bieten unterschiedliche Möglichkeiten, in den Arbeitsprozess von Ansible eingebunden zu werden. Neben den Variablen können auch Kontrollstrukturen wie Tests und Schleifen mit Ansible verwendet und mit Variablen kombiniert werden.

### 15.4.1 Variablen im Inventory



## Variablen im Inventory

- Hostvariablen
- Gruppenvariablen

**Verbindungsvariablen für einen einzelnen Host**

```
<hostname> ansible_host=172.20.1.121 ansible_port=2222
```

**Verbindungsvariablen für eine Gruppe**

```
[<groupname>:vars]
ansible_user=tux
ansible_port=2222
```

**Eine Gruppe aus Gruppen erstellen:**

```
[name_der_neuen_gruppe:children]
Gruppe1
Gruppe2
```

---

B1 Systems GmbH      Terraform & Ansible Grundlagen      162 / 279

Ansible kann Variablen auslesen, die im Inventory deklariert sind. Variablen können für einzelne Hosts oder für Gruppen gelten.

#### Hostvariablen

Für einzelne Hosts können z. B. alternative Namen festgelegt werden, beispielsweise falls der Host ausschließlich über eine IP-Adresse erreichbar ist:

```
<hostname> ansible_host=172.20.1.121
```

Somit wäre der Host mit der IP-Adresse 172.20.1.121 unter dem Hostnamen `<hostname>` in einem Ad-Hoc-Befehl oder Playbook ansprechbar.

Weitere häufig verwendete Verbindungsvariablen sind:

```
ansible_port=
```

und

```
ansible_user=
```

## 15 Ansible Playbooks

mit denen Sie den SSH-Port sowie den SSH-Benutzer für den Host festlegen können. Der Eintrag im Inventory sähe dann z. B. so aus:

```
tuxhost ansible_host=172.20.1.121 ansible_port=2222 ansible_user=tux
```

Für die Deklaration solcher Hostvariablen hält Ansible eine Reihe an Umgebungsparametern bereit, mit denen die einzelnen Hosts definiert werden können. Eine Liste der verfügbaren Parameter ist einsehbar unter: [https://docs.ansible.com/ansible/latest/user\\_guide/intro\\_inventory.html#connecting-to-hosts-behavioral-inventory-parameters](https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html#connecting-to-hosts-behavioral-inventory-parameters)

Zusätzlich zu den vorgegebenen Verbindungsvariablen können Sie im Inventory auch selber Variablen deklarieren, was allerdings nicht empfohlen wird. Das Deklarieren eigener Variablen erfolgt nach dem gleichen Muster:

```
host variable=<Wert_der_Variable> weitere_variable=<Wert>
```

Zum Beispiel:

```
host http_port=80 maxRequestsPerChild=808
```



### Hinweis Ungruppierte Hosts im Inventory

Da Gruppedefinitionen im Inventory nur enden, wenn eine neue Gruppe definiert wird, müssen ungruppierte Hosts vor Beginn des Gruppenteils im Inventory stehen. Mit der Host-Angabe „ungrouped“ können in einem Ad-Hoc-Befehl oder in einem Playbook alle Hosts im Inventory referenziert werden, die keiner Gruppe zugeordnet wurden.

## Gruppenvariablen

Neben den einzelnen Hostvariablen können im Inventory auch Gruppenvariablen deklariert werden. Mit :children können Sie vorhandene Gruppen in einer neuen Gruppe zusammenfassen:

```
[name_der_neuen_gruppe:children]
Gruppe1
Gruppe2
```

Zum Beispiel:

```
[ubuntu]
ubuntu1
ubuntu2

[suse]
suse1
suse2

[ubuntususe:children]
ubuntu
suse
```

Somit würden in einem Ad-Hoc-Befehl oder in einem Playbook mit dem Aufruf von `ubuntususe` die beiden Gruppen `ubuntu` und `suse` aufgerufen werden. Die so zusammengefassten Gruppen lassen sich wie normale Gruppen behandeln und nach dem selben Muster mit anderen Gruppen zu weiteren Gruppen zusammenfassen:

```
[ubuntu]
ubuntu1
ubuntu2

[suse]
suse1
suse2

[ubuntususe:children]
ubuntu
suse

[webserver:children]
ubuntususe
somegroup
```

Es können nach folgendem Prinzip auch Variablen für Gruppen deklariert werden:

```
[ubuntu]
ubuntu1
ubuntu2

[ubuntu:vars]
ansible_user=tux
Variable=Wert
```

Die so gesetzten Gruppenvariablen gelten für jeden Host der entsprechenden Gruppe. Für Kindgruppen gilt das gleiche Muster:

```
[ubuntu]
ubuntu1
ubuntu2

[suse]
suse1
suse2

[ubuntususe:children]
ubuntu
suse

[ubuntususe:vars]
ansible_user=tux
Variable=Wert
```



#### 15.4.2 Aufgabenteil



## Aufgabenteil – Variablen im Inventory

- ① Deklarieren Sie eine Variable im Inventory, mit der das System centos2 unter dem Hostnamen b1 und dem Port 2222 erreichbar ist.
- ② Erstellen Sie eine neue Gruppe namens webserver, die die Systeme ubuntu1 und suse1 beinhaltet. Erstellen Sie ferner eine Gruppe deployment, die die Systeme centos1 und suse2 beinhaltet. Erstellen sie nun eine Gruppe productive, in der die beiden Gruppen webserver und deployment zusammengefasst sind.
- ③ Deklarieren Sie die Variablen http\_default\_port: 80 und http\_webroot: /var/www/web für die Gruppe productive.

1. Deklarieren Sie eine Variable im Inventory, mit der das System centos2 unter dem Hostnamen b1 und dem Port 2222 erreichbar ist.
2. Erstellen Sie eine neue Gruppe namens webserver, die die Systeme ubuntu1 und suse1 beinhaltet. Erstellen Sie ferner eine Gruppe deployment, die die Systeme centos1 und suse2 beinhaltet. Erstellen sie nun eine Gruppe productive, in der die beiden Gruppen webserver und deployment zusammengefasst sind.
3. Deklarieren Sie die Variablen http\_default\_port: 80 und http\_webroot: /var/www/web für die Gruppe productive.

## Musterlösung

1. Deklarieren Sie eine Variable im Inventory, mit der das System **centos2** unter dem Hostnamen **b1** und dem Port **2222** erreichbar ist:

Eintrag in `inventory_exercise`:

```
b1 ansible_host=centos2 ansible_port=2222
```

2. Erstellen Sie eine neue Gruppe namens **webserver**, die die Systeme **ubuntul** und **suse1** beinhaltet. Erstellen Sie ferner eine Gruppe **deployment**, die die Systeme **centos1** und **suse2** beinhaltet. Erstellen sie nun eine Gruppe **productive**, in der die beiden Gruppen **webserver** und **deployment** zusammengefasst sind:

Weitere Einträge in `inventory_exercise`:

```
[webserver]
ubuntul
suse1
```

```
[deployment]
centos1
suse2
```

```
[productive:children]
webserver
deployment
```

3. Deklarieren Sie die Variablen `http_default_port=80` und `http_webroot=/var/www/web` für die Gruppe **productive**.

Weitere Einträge in `inventory_exercise`:

```
[webserver]
ubuntul
suse1
```

```
[deployment]
centos1
suse2
```

```
[productive:children]
webserver
deployment
```

```
[productive:vars]
http_default_port=80
http_webroot=/var/www/web/
```

### 15.4.3 host\_vars und group\_vars



## host\_vars und group\_vars

### Variablen in spezieller Struktur

```
./inventory
./host_vars/
./host_vars/host1.yml
./group_vars/
./group_vars/webserver/
./group_vars/webserver/website.yml
./group_vars/webserver/allgemein.yml
```

B1 Systems GmbH
Terraform & Ansible Grundlagen
164 / 279

Statt die Variablen für Hosts und Gruppen direkt in das Inventory zu schreiben, bietet es sich an, die Variablen in eigene Dateien auszulagern. Ansible sieht dafür die Dateien `host_vars` und `group_vars` vor. Es gibt eine Standardreihenfolge, in der mögliche Variablendateien eingelesen werden. Dabei ist zu beachten, dass immer die zuletzt eingelesene Variablendefinition gültig ist, da sie vorangegangene überschreibt.

Ansible sucht zunächst im Pfad des verwendeten Inventory nach `host_vars-` und `group_vars`-Dateien:

```
./inventory
./host_vars/
./host_vars/host1.yml
./group_vars/
./group_vars/webserver/
./group_vars/webserver/website.yml
./group_vars/webserver/allgemein.yml
```

Ausgehend vom Verzeichnis, in dem das angegebene Inventory liegt, sucht Ansible nach bestimmten Verzeichnissen und Dateien, in denen Variablen deklariert werden können. Sollten Inventory und Playbook in verschiedenen Verzeichnissen liegen, so sucht Ansible auch im Playbook-Verzeichnis in der selben Reihenfolge wie im Inventory-Verzeichnis.

## 15 Ansible Playbooks

Das Playbook-Verzeichnis wird allerdings *nach* dem Inventory-Verzeichnis eingelesen, sodass auch hier wieder zu beachten ist, dass die zuletzt eingelesenen Variablen vorangegangene Deklarationen mit identischer Bezeichnung überschreiben.

Die `host_vars`- und `group_vars`-Dateien werden, wie auch die Playbooks, in der YAML-Syntax verfasst.

### **host\_vars**

Die Dateien für ungruppierte Hosts werden nach den Hosts selber, also `<host>.yml`, benannt und im Verzeichnis `host_vars` abgelegt.

Gibt es dort eine Datei `all.yml`, so gilt sie für alle ungruppierten Hosts.

Hat die Datei den selben Namen wie ein Host, so gelten die darin deklarierten Variablen ausschließlich für diesen Host.

Variablen für den im Inventory eingetragenen Host `webserver_api` würden Sie wie folgt in die Datei `host_vars/webserver_api.yml` eintragen:

```
---
```

```
ansible_host: <hostname>
ansible_port: 2222
```

### **group\_vars**

Die Dateien mit der Deklaration von Variablen für Gruppen von Hosts sollten im Unterverzeichnis `group_vars` abgelegt werden. Auch hier ist es wieder möglich, eine Datei `all.yml` zu erstellen, deren Variablen-deklaration dann für alle im Inventory eingetragenen Zielsysteme gilt.

Bei einer nach einer Gruppe benannten Datei gilt deren Inhalt nur für die entsprechende Gruppe.

Ferner ist es möglich, innerhalb von `group_vars` nach den Gruppen benannte Unterverzeichnisse zu erstellen.

In den Verzeichnissen, die nach Gruppen benannt sind, ist es nicht nötig, eine `all.yml` zu erstellen, da alle Dateien in diesen Verzeichnissen automatisch für die gesamte Gruppe gelten, nach der das Verzeichnis benannt ist.

Variablen für alle im Inventory eingetragenen Zielsysteme werden in der Datei `all.yml` direkt im Verzeichnis `group_vars` eingetragen, z. B.:

```
---
```

```
ssh_port: 22
admin_user: tux
```

Um Variablen für eine bestimmte Gruppe zu deklarieren, muss die Datei nach der Gruppe benannt sein. Im folgenden Beispiel ist es `group_vars/webserver.yml` für die Gruppe „`webserver`“:

```
---
webserver:
  port: 8080
  webroot: /var/www

database:
  user: admin
  port: 3306

path_env: /opt/dev

database_host: 127.0.0.1
database_name: app
database_user: sql

ssl: true
ev_ssl: false
```

Es kann ebenfalls ein Verzeichnis mit dem Namen der Gruppe erstellt werden, in das Variablen dateien gelegt werden können, die dann für die gesamte Gruppe gelten:

```
# mkdir group_vars/webserver
```

Alle Dateien in diesem Verzeichnis werden von Ansible nach Variablen für die Gruppe webserver durchsucht, unabhängig von deren Bezeichnung:

```
# cat group_vars/webserver/funny_variables.yml
---
httpd_package_state: latest
apache_vhosts:
  - name: b1.de
    docroot: '/var/www/b1.de/'
  - name: training.b1.de
    docroot: '/var/www/training.b1.de/'

jinjaclist:
  - "the first point is the best"
  - "this could be a value"
  - "remember the field index"
  - "so this is 3, right?"
  - "4-5"
```

### Variablen für alle ungruppierten Hosts

Mit dem Erstellen einer Datei ungrouped.yml im Verzeichnis group\_vars ist es möglich, Variablen für alle Zielsysteme zu deklarieren, die keiner Gruppe zugeordnet sind.



#### 15.4.4 Aufgabenteil



## Aufgabenteil – host\_vars und group\_vars

Überführen Sie die Variablen aus dem Inventory in die passenden host\_vars- und group\_vars-Dateien.

Auszug aus dem Inventory:

[...]

```
[productive:children]
webserver
deployment

[productive:vars]
http_default_port=80
http_webroot=/var/www/web/
```

Überführen Sie die Variablen aus dem Inventory der Übungsaufgabe von S.300 in die passenden host\_vars- und group\_vars-Dateien.

Das Inventory:

```
[webserver]
ubuntul
suse1

[deployment]
centos1
suse2

[productive:children]
webserver
deployment

[productive:vars]
http_default_port=80
http_webroot=/var/www/web/
```

## Musterlösung

**Überführen Sie die Variablen aus dem Inventory der Übungsaufgabe von S.300 in die passenden `host_vars`- und `group_vars`-Dateien:**

Das Inventory:

```
[webserver]
ubuntul
suse1

[deployment]
centos1
suse2

[productive:children]
webserver
deployment

[productive:vars]
http_default_port=80
http_webroot: /var/www/web/
```

Neue Datei `group_vars/productive.yml`:

```
---
http_default_port: 80
http_webroot: /var/www/web/
```

#### 15.4.5 Ansible Facts



## Ansible Facts

- Facts werden durch das Modul `setup` abgerufen
  - wird bei jedem Aufruf eines Playbooks automatisch ausgeführt
  - alle Facts stehen im Playbook zur Verfügung
- Ansible hat eine Reihe vordefinierter Facts
- können wie Variablen ausgegeben und eingesetzt werden
- können mit `gather_facts: no|false` ausgeschaltet werden

### Ausgabe von Facts mit dem `ansible.builtin.setup`-Modul

```
# ansible ubuntu1 -i inventory -m ansible.builtin.setup
```

Bei den `ansible_facts` handelt es sich um von Ansible abgerufene Informationen über die Hostsysteme, die als vordefinierte Variablen zur Verfügung gestellt werden. Diese Informationen können Sie in Ad-Hoc Befehlen mit dem `setup`-Modul ausgeben lassen:

```
# ansible ubuntu1 -i inventory -m ansible.builtin.setup
```

Die `ansible_facts` werden bei jedem Aufruf des Befehls `ansible-playbook` automatisch gesammelt und von Ansible im Playbook als vordefinierte Variablen zu Verfügung gestellt. Diese Variablen können Sie sich in einem Playbook mit dem `debug`-Modul ausgeben lassen:

```
---
- name: get ansible facts with the debug modul
  hosts: ubuntu1
  tasks:
    - name: the debug modul
      ansible.builtin.debug:
        var: ansible_facts
```

## 15 Ansible Playbooks

Dies ergibt:

```
# ansible-playbook -i /root/code/inventory facts.yml

PLAY [get ansible facts with the debug modul] **

TASK [Gathering Facts] **
ok: [ubuntul]

TASK [the debug modul] **
ok: [ubuntul] => {
    "ansible_facts": {
        "all_ipv4_addresses": [
            "172.20.1.121"
        ],
        "all_ipv6_addresses": [
            "fe80::42:acff:fe14:179"
        ],
        "ansible_local": {},
        "apparmor": {
            "status": "disabled"
        },
    [...]
```

So wie im Ad-Hoc Befehl mit dem Parameter `filter` nach bestimmten Facts gesucht werden kann, können Sie auch im Playbook auf bestimmte Facts zugreifen:

```
---
- name: get ansible facts with the debug modul
  hosts: ubuntul
  tasks:
    - name: the debug modul
      ansible.builtin.debug:
        var: ansible_all_ipv4_addresses
```

Das debug-Modul gibt nun nur die gefilterten Facts wieder:

```
# ansible-playbook -i /root/code/inventory facts.yml

PLAY [get ansible facts with the debug modul] **

TASK [Gathering Facts] **
ok: [ubuntul]

TASK [the debug modul] **
ok: [ubuntul] => {
    "ansible_all_ipv4_addresses": [
        "172.20.1.121"
    ]
}

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



# Ansible Facts

Auszug aus den Ansible Facts des setup-Moduls:

```
# ansible ubuntu1 -i inventory -m ansible.builtin.setup

ubuntu1 | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "172.20.1.121"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::42:acff:fe14:179"
        ],
        "ansible_apparmor": {
            "status": "disabled"
        },
        [...]
```

Anstatt sich die Werte der Facts einfach nur ausgeben zu lassen, können Sie diese direkt als Variablen aufrufen und verwenden. Damit Variablen im Playbook aufgerufen werden können, müssen diese in ein Paar geschweifter Klammern gesetzt werden:

```
"{{ variable }}"
```

In einem Playbook können die Facts dann z. B. so als Variablen aufgerufen werden:

```
---
- name: show ansible facts with the debug modul
  hosts: ubuntu1
  tasks:
    - name: the debug modul
      ansible.builtin.debug:
        msg: "Dieses System hat den Namen {{ ansible_hostname }} und
              die Kernelversion {{ ansible_kernel }}."
```

Beim Ausführen des Playbooks ruft das debug-Modul die Variablen ab und gibt deren Werte mit dem Parameter msg aus:

```
# ansible-playbook -i /root/code/inventory /root/code/debug_vars.yml

PLAY [show ansible facts with the debug modul] **
```

## 15 Ansible Playbooks

```
TASK [Gathering Facts] **
ok: [ubuntul]

TASK [the debug modul] **
ok: [ubuntul] => {
    "msg": "Dieses System hat den Namen ubuntul und die Kernelversion
        ↪ 5.0.10-200.fc29.x86_64."
}

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

### **gather\_subset**

Mit der Option `gather_subset` können Sie die Facts, die von `ansible-playbook` gesammelt werden, auf einzelne oder eine bestimmte Anzahl von Unterkategorien beschränken:

```
---
- name: a playbook to gather_subset(s)
  hosts: ubuntul
  gather_subset: network
  tasks:
```

Mehrere Subsets werden als Liste angegeben:

```
---
- name: a playbook to gather_subset(s)
  hosts: ubuntul
  gather_subset:
    - network
    - ohai
  tasks:
```

Die Parameter für die Option `gather_subset`:

SAMMELT FACTS ÜBER:	SAMMELT ALLE FACTS AUSSER FÜR:
all	!all
min	!min
network	!network
hardware	!hardware
virtual	!virtual
facter	!facter
ohai	!ohai

**gather\_facts**

Soll ansible-playbook gänzlich auf das Einsammeln der Facts verzichten, beispielsweise um Zeit zu sparen, können Sie dies mit der Option `gather_facts` mit dem Argument `no` deaktivieren:

```
---
```

```
- name: deactivate fact gathering
  hosts: ubuntu1
  gather_facts: no
```

Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de



#### 15.4.6 Variablen im Playbook

## Variablen im Playbook

- vars: im Playbook
- YAML-Schreibweise
- registrierte Variablen

Variablen können direkt im Playbook deklariert werden:

```
- hosts: productive
  vars:
    http_port: 80
```

Beispiel eines Aufrufs einer Variable mit Jinja2-Syntax:

```
- name: get me the port number
  ansible.builtin.debug:
    msg: "Der Port hat die Nummer {{ http_port }}."
```

Variablen können auch direkt im Playbook deklariert und in Tasks eingebunden werden. Dabei überschreiben die in Playbooks deklarierten Variablen alle Variablen, die möglicherweise vorher im Inventory, einer host\_vars- oder group\_vars-Datei deklariert wurden.

Beim Deklarieren der Variablen müssen Sie darauf achten, gültige Variablennamen zu erzeugen. Gültige Variablen dürfen Buchstaben, Ziffern und Unterstriche enthalten, jedoch nicht mit einer Ziffer beginnen. Gültige Variablen wären z. B.:

```
hostname
hostname3
host_name
```

Die Variablen weisen Sie im Playbook mit dem Parameter vars: zu. Wie im Playbook üblich wird dafür die YAML-Syntax und das Key:Value Prinzip benutzt:

```
---
- name: just a playbook with a variable
  hosts: centos
  vars:
    http_port: 80
```

Wie die `ansible_facts` werden auch die im Playbook deklarierten Variablen mit geschweiften Klammern aufgerufen:

```
---
- name: just a playbook with one variable
  hosts: centos1
  vars:
    http_port: 80
  tasks:
    - name: A task where the variable will be called
      ansible.builtin.debug:
        msg: "Der Port für http lautet: {{ http_port }}"
```

Die Ausgabe von `ansible-playbook`:

```
# ansible-playbook -i /root/code/inventory
  ↳ /root/code/variable_chapter.yml

PLAY [just a playbook with a variable] **

TASK [Gathering Facts] **
ok: [centos1]

TASK [A task where the variable will be called] **
ok: [centos1] => {
    "msg": "Der Port für http lautet: 80"
}

PLAY RECAP **
centos1 : ok=2     changed=0     unreachable=0     failed=0
```

### 15.4.7 Dictionaries & Listen in YAML



## YAML

### Dictionaries:

```
---
vars:
  ports:
    http_port: 80
    https_port: 443
    database_port: 3306
```

### Liste in einem Dictionary:

```
vars:
  names:
    - tux
    - tix
    - tax
```

## Dictionaries in YAML

Einträge in YAML-Dateien – und damit auch Playbooks – sind nach dem **Schlüssel: Wert**-Prinzip aufgebaut. Diese Einträge bezeichnet man bei YAML als *Dictionary*.

Ein Playbook enthält eine Liste von einzelnen *Plays*, bei denen es sich um *Dictionaries* handelt. Für das Deklarieren von Variablen bedeutet dies, dass auch diese in Dictionaries angelegt werden können. Im oben aufgeführten Variablenbeispiel handelt es sich bei dem Variablenteil um ein eigenes Dictionary:

```
vars:
  http_port: 80
```

In diesem Dictionary `vars` gibt es einen Eintrag, in dem über den Schlüssel `http_port` der Wert 80 deklariert wird. Sollen mehrere Schlüssel mit Werten versehen werden, also mehrere Variablen deklariert, so lässt sich das Dictionary erweitern:

```
vars:
  ports:
    http_port: 80
```

```
https_port: 443
database_port: 3306
```

Hier bekommt das Dictionary `vars` den Eintrag `ports`, in dem einer Reihe von Schlüsseln Werte zugeordnet werden. Beim Eintrag `ports` selber handelt es sich ebenfalls um ein Dictionary.

In einem Playbook können diese Paare aus Schlüsseln und Werten nun als Variablen abgerufen werden:

```
---
- name: just a playbook with three variables
  hosts: centos1
  vars:
    ports:
      http_port: 80
      https_port: 443
      database_port: 3306
```

## Listen in YAML

Ein weiterer in YAML verwendeter Datentyp sind Listen. Listen enthalten in einem Key nicht direkt Key-Value Paare, sondern mit – definierte Listenpunkte:

Liste:

- "erster Punkt"
- "zweiter Punkt"
- "dritter Punkt"

In diesem Beispiel wird unter dem Key `Liste` eine Liste mit drei Listeneinträgen definiert.

## Dictionaries und Listen in YAML

Die beiden Datentypen Dictionaries und Listen lassen sich in YAML beliebig kombinieren. So kann beispielsweise ein Listenpunkt Key-Value-Paare oder Dictionaries enthalten, wie es im folgenden Beispiel für ein Play der Fall ist:

```
---
- name: erstes Play
  hosts: all
  vars:
    my_var:
      key: value
```

Beim Erstellen von Plays in Playbooks wird zunächst ein Listenpunkt erzeugt, der dann die Key-Value-Paare mit den Keys `name:` und `hosts:` enthält. Im Listenpunkt wird jedoch auch das Dictionary `vars:` eingetragen, das wiederum das Dictionary `my_var:` enthält, in dem das Key-Value-Paar `key: value` eingetragen ist.

## 15 Ansible Playbooks

Dictionaries können neben Key-Value-Paaren auch Listen enthalten:

```
---
```

```
- name: erstes Play
  hosts: all
  vars:
    names:
      - tux
      - tix
      - tax
```

In diesem Beispiel enthält das Dictionary `vars:` einen Key `names:`, der wiederum drei Listenpunkte (`tux`, `tix`, `tax`) beinhaltet. Diese Listenpunkte könnten ihrerseits wieder Dictionaries enthalten und diese wieder Listen und immer so weiter. Dictionaries und Listen sind in YAML beliebig verschachtelbar.



# Referenzierung von Variablen aus Dictionaries

## Dictionary vars: mit Variablen

```
---
vars:
  ports:
    http_port: 80
    https_port: 443
```

## Referenzierung einer Variable aus einem Dictionary

```
tasks:
  - name: A task where the variables will be called
    ansible.builtin.debug:
      msg: "Der Port lautet: {{ ports['http_port'] }}.

● alternative Schreibweise: ports.http_port
```

Um Variablen in Dictionary-Strukturen zu referenzieren, bietet Ansible zwei Schreibweisen. Bei der ersten werden die Einträge mit Punkten getrennt angegeben:

`ports.http_port`

Bei der anderen Schreibweise werden die Einträge in eckigen Klammern eingefasst angegeben:

`ports['http_port']`

Beide Einträge verweisen identisch auf den Eintrag `http_port` im Dictionary `ports`. Da es bei der „punktierter“ Schreibweise zu Konflikten mit der Syntax von Python kommen kann, ist die Schreibweise mit den Klammern empfohlen und wird auch in den folgenden Beispielen verwendet.

In einem Playbook würden die im Dictionary deklarierten Variablen wie folgt aufgerufen:

```
---
- name: just a playbook with three variables
  hosts: centos1
  vars:
    ports:
      http_port: 80
      https_port: 443
```

## 15 Ansible Playbooks

```
    database_port: 3306
tasks:
  - name: A task where the variables will be called
    ansible.builtin.debug:
      msg: "Der Port für http lautet: {{ ports['http_port'] }}, der
            ↪ für https lautet: {{ ports['https_port'] }} und der
            ↪ Datenbankport lautet: {{ ports['database_port'] }}."
```

### Ausführung des Playbooks:

```
# ansible-playbook -i /root/code/inventory
  ↪ /root/code/variable_chapter.yml

PLAY [just a playbook with three variables] **

TASK [Gathering Facts] **
ok: [centos1]

TASK [A task where the variables will be called] **
ok: [centos1] => {
  "msg": "Der Port für http lautet: 80, der für https lautet: 443
         ↪ und der Datenbankport lautet: 3306."
}

PLAY RECAP **
centos1 : ok=2     changed=0     unreachable=0     failed=0
```

Es ist ebenfalls möglich, in einem Dictionary eine Variable mit einer Liste von Werten zu deklarieren:

```
---
- name: just a list in a variable
  hosts: suse2
  vars:
    names:
      - tux
      - tix
      - tax
```

Innerhalb der Liste könnte wieder ein Dictionary angelegt werden:

```
---
- name: just a list in a variable
  hosts: suse2
  vars:
    names:
      - tux:
          food: pasta
          color: green
          os: linux
      - tix
      - tax
```

Diese Kombination von Dictionaries und Listen ließe sich beliebig fortsetzen.



**Hinweis** *Alternative Schreibweise für YAML-Syntax*

Die YAML-Syntax verfügt zusätzlich über eine kompakte Schreibweise, in der ein gesamtes Dictionary in eine einzige Zeile geschrieben werden kann. In bestimmten Situationen, z. B. der Kombination mit `loop`, kann diese Schreibweise nützlich sein. Da die ausführliche Schreibweise eine bessere Übersichtlichkeit gewährleistet, wird diese in dieser Unterlage bevorzugt.

## Listen – Schreibweisen

### Kompakte Form

```
users: [ 'peter', 'paul', 'mathilda' ]
```

### Ausführliche Form

```
users:
  - 'peter'
  - 'paul'
  - 'mathilda'
```

### Referenzierung eines Listenpunktes:

```
tasks:
  - name: get the first list item
    ansible.builtin.debug:
      msg: "The first name is: {{ users[0] }}."
```

Auch für Listen bietet YAML verschiedene Schreibweisen.

Ein Beispiel für die kompakte YAML-Schreibweise:

```
Dictionary: { "Schlüssel": "Wert" , "liste": [ "1. Punkt", "2.
  ↪ Punkt"], "dictionary": { "1. Eintrag": "1. Wert", "2. Eintrag":
  ↪ "2.Wert" } }
```

In der kompakten Schreibweise werden Listen in eckigen Klammern [] eingefasst und Dictionaries in geschweifte {}.

Dieser Eintrag entspricht folgendem Beispiel in der „ausführlichen“ Schreibweise:

```
Dictionary:
  Schlüssel: Wert
  liste:
    - 1. Punkt
    - 2. Punkt
  dictionary:
    1. Eintrag: 1. Wert
    2. Eintrag: 2. Wert
```

#### 15.4.8 Registrierte Variablen



## Registrierte Variablen

- werden nicht vor Ausführen des Playbooks gesetzt, sondern während der Ausführung
- können Resultate aus Tasks als Variable deklarieren:  
register: <varname>
- lassen sich wie normale Variablen aufrufen
- sind innerhalb eines Playbooks gültig

### Task mit register:

```
- name: first task to register some output
  ansible.builtin.command: "id"
  register: id_register
```

Bei *registrierten Variablen* handelt es sich um Variablen, in denen das Ergebnis eines ausgeführten Befehls oder Prozesses hinterlegt wird. Im Gegensatz zu nicht-registrierten Variablen muss hier der Wert also nicht vor dem Ausführen des Playbooks gesetzt werden, sondern er wird während der Ausführung registriert:

```
- name: lets register some variables
  hosts: centos2
  vars:
    cmd: id
  tasks:
    - name: first task to register some output
      ansible.builtin.command: "{{ cmd }}"
      register: id_register
    - name: second task to display the results
      ansible.builtin.debug:
        msg: "The command {{ cmd }} was executed successfully. It
              ↪ returned {{ id_register['rc'] }} and logged {{
              ↪ id_register['stdout'] }} to STDOUT"
```

## 15 Ansible Playbooks

Bei Ausführung des Playbooks:

```
# ansible-playbook -i /root/code/inventory /root/code/register.yml

PLAY [lets register some variables] **

TASK [Gathering Facts] **
ok: [centos2]

TASK [first task to register some output] **
changed: [centos2]

TASK [second task to display the results] **
ok: [centos2] => {
    "msg": "The command id was executed successfully. It returned 0
           ↪ and logged uid=0(root) gid=0(root) groups=0(root) to STDOUT"
}

PLAY RECAP **
centos2 : ok=3     changed=1     unreachable=0     failed=0
```

In diesem Beispiel wird das Ergebnis des Befehls `id`, welcher in der Variable `cmd` deklariert wird, als Variable `id_register` registriert. Die vollständige Ausgabe des Registers `id_register` sähe wie folgt aus:

```
" {'stderr_lines': [], u'changed': True, u'end': u'2019-05-17
    ↪ 06:59:05.260976', 'failed': False, u'stdout': u'uid=0(root)
    ↪ gid=0(root) groups=0(root)', u'cmd': u'id', u'rc': 0, u'start':
    ↪ u'2019-05-17 06:59:05.059532', u'stderr': u'', u'delta':
    ↪ u'0:00:00.201444', 'stdout_lines': [u'uid=0(root) gid=0(root)
    ↪ groups=0(root)']}"
```

Aus dieser Ausgabe werden nun im zweiten Task der Return Code sowie STDOUT aus der registrierten Variable referenziert und mit dem debug-Modul ausgegeben. Im Register wurde also das Ergebnis des ersten Tasks gespeichert, oder registriert, um im Playbook als Variable zu dienen.

### 15.4.9 Aufgabenteil



## Aufgabenteil – Variablen im Playbook

- ① Deklarieren Sie in einem Playbook drei Variablen und lassen Sie diese in einem Task mit dem `ansible.builtin.debug`-Modul und dem Parameter `msg` ausgeben.
- ② Schreiben Sie ein Playbook, in dem Sie in einem Task mit dem Modul `ansible.builtin.apt_repository` sicherstellen, dass das Repository `ppa:nginx/stable` auf dem System `ubuntu2` vorhanden ist. Registrieren Sie das Ergebnis in einer Variablen und lassen Sie den Wert des Schlüssels `state` aus dieser Variablen in einem weiteren Task mit dem `ansible.builtin.debug`-Modul ausgeben.

1. Deklarieren Sie in einem Playbook drei Variablen und lassen Sie diese in einem Task mit dem `ansible.builtin.debug`-Modul und dem Parameter `msg` ausgeben.
2. Schreiben Sie ein Playbook, in dem Sie in einem Task mit dem Modul `ansible.builtin.apt_repository` sicherstellen, dass das Repository `ppa:nginx/stable` auf dem System `ubuntu2` vorhanden ist. Registrieren Sie das Ergebnis in einer Variablen und lassen Sie den Wert des Schlüssels `state` aus dieser Variablen in einem weiteren Task mit dem `ansible.builtin.debug`-Modul ausgeben.

## Musterlösung

- Deklarieren Sie in einem Playbook drei Variablen und lassen Sie diese in einem Task mit dem `ansible.builtin.debug`-Modul und dem Parameter `msg` ausgeben:

```
---
- name: a playbook where a variable will be declared and debugged
  hosts: ubuntu1
  vars:
    name1: tux
    os: linux
    company: b1
  tasks:
    - name: a task with a debug module
      ansible.builtin.debug:
        msg: "My name is {{ name1 }}, I love {{ os }} and I work
              → at {{ company }}."
```

Wird das Playbook ausgeführt, gibt das `debug`-Modul die in Variablen deklarierten Werte aus:

```
# ansible-playbook -i /root/code/inventory
  → /root/code/variable_exercise.yml

PLAY [a playbook where a variable will be declared and debugged]
  → **

TASK [Gathering Facts] **
ok: [ubuntu1]

TASK [a task with a debug module] **
ok: [ubuntu1] => {
    "msg": "My name is tux, I love linux and I work at b1."
}

PLAY RECAP **
ubuntu1 : ok=2      changed=0      unreachable=0      failed=0
```

- Schreiben Sie ein Playbook, in dem Sie in einem Task mit dem Modul `ansible.builtin.apt_repository` sicherstellen, dass das Repository `ppa:nginx/stable` auf dem System `ubuntu2` vorhanden ist. Registrieren Sie das Ergebnis in einer Variablen und lassen Sie den Wert des Schlüssels `state` aus dieser Variablen in einem weiteren Task mit dem `ansible.builtin.debug`-Modul ausgeben:

Das Playbook könnte wie folgt aussehen:

```
---
- name: make sure the nginx repository is present on the ubuntu2
  → machine
  hosts: ubuntu2
  tasks:
```

```

- name: make sure the repository is present
  ansible.builtin.apt_repository:
    repo: ppa:nginx/stable
    state: present
  register: ppa_register

- name: getting the info from the registered Variable to see
  ↪ if the repository is present
  ansible.builtin.debug:
    msg: " The repository {{ ppa_register['repo'] }} is {{
      ↪ ppa_register['state'] }} on this system."

```

Ausgeführt gibt das debug-Modul die Werte für das Repository und den Installationsstatus auf dem System aus:

```

# ansible-playbook -i /root/code/inventory
  ↪ /root/code/ppa_register.yml

PLAY [make sure the nginx repository is present on the ubuntu2
  ↪ machine] **

TASK [Gathering Facts] **
ok: [ubuntu2]

TASK [make sure the repository is present] **
changed: [ubuntu2]

TASK [getting the info from the registered Variable to see if the
  ↪ repository is present] **
ok: [ubuntu2] => {
  "msg": " The repository ppa:nginx/stable is present on this
    ↪ system."
}

PLAY RECAP **
ubuntu2 : ok=3    changed=1    unreachable=0    failed=0

```

#### 15.4.10 Variablen auf der Kommandozeile



## Variablen auf der Kommandozeile

- Deklaration auf der Kommandozeile mit `--extra-vars`, kurz: `-e`
- werden als „*Extra-Vars*“ bezeichnet
- überschreiben alle anderen Variablen
- können mit `vars_files` genutzt werden

Den Wert der Variablen `homedir` mit einer Extra-Var überschreiben:

```
# ansible-playbook -i inventory vars.yml --extra-vars "homedir=/home/tix/"
```

Extra-Vars aus einer Datei übergeben:

```
# ansible-playbook -i inventory vars.yml -e @/root/code/extravars_vars.yml
```

Mit den *Extra-Vars* bietet Ansible eine Möglichkeit, Variablen direkt auf der Kommandozeile zu deklarieren und an die Befehle `ansible` oder `ansible-playbook` zu übergeben. Auf diese Art deklarierte Variablen überschreiben Variablen, die an anderer Stelle der Ansible-Struktur definiert wurden. Dies kann sinnvoll sein, wenn bestimmte Werte getestet werden sollen oder der Befehl einmalig mit bestimmten Werten ausgeführt werden soll. Abzuraten ist es dagegen, die Extra-Vars als dauerhafte Lösung im Produktivbetrieb zu betrachten.

Gesetzt werden die Extra-Vars mit der Option `--extra-vars` bzw. `-e` direkt auf der Kommandozeile. Dabei erfolgt die Deklaration mit der Syntax `key=value`, entsprechend der Key: Value Schreibweise im YAML-Format.

### Variablen in Ad-Hoc Befehlen

Mit der Verwendung von Extra-Vars ist es möglich, Variablen in Ad-Hoc Befehlen zu nutzen. Anstelle eines Strings wird dann dem Modul als Argument die Variable mitgegeben. Die Variable selbst wird unter Angabe der Option `--extra-vars` deklariert:

```
# ansible ubuntu1 -i ../../inventory -m ansible.builtin.file -a "path={{ homedir }}/foo state=directory" --extra-vars "homedir=/home/tux"
```

```
ubuntul | SUCCESS => {
    "changed": false,
    "gid": 0,
    "group": "root",
    "mode": "0755",
    "owner": "root",
    "path": "/home/tux/foo",
    "size": 4096,
    "state": "directory",
    "uid": 0
}
```

In diesem Beispiel wird dem Argument path des Moduls `file` die Variable `homedir` als Zielpfad übergeben. Die Variable `homedir` wird als `/home/tux` deklariert und im Folgenden von `ansible` korrekt aufgelöst.

Es können auf diese Art auch mehrere Variablen auf einmal deklariert und eingesetzt werden. Dafür werden die Variablen einfach mit einem Leerzeichen getrennt angegeben:

```
# ansible ubuntul -i ../inventory -m ansible.builtin.file -a "path={{ homedir }}/{{ folder }} state=directory" --extra-vars
  "homedir=/home/tux folder=foo"
ubuntul | SUCCESS => {
    "changed": false,
    "gid": 0,
    "group": "root",
    "mode": "0755",
    "owner": "root",
    "path": "/home/tux/foo",
    "size": 4096,
    "state": "directory",
    "uid": 0
}
```

Hier wird zusätzlich zur Variablen `homedir` noch die Variable `folder` deklariert. Das Argument `path` bekommt nun die beiden Variablen als `{{ homedir }}/{{ folder }}` übergeben und `ansible` löst diese zu `/home/tux/foo` auf.

## Extra-Vars mit Playbooks

Der Nutzen von Variablen in Ad-Hoc Befehlen ist eher gering, da diese zumeist mit Argumenten aufgerufen werden und die Deklaration von Variablen einen zusätzlichen Aufwand bedeuten würde. Eine plausible Anwendung liegt jedoch im einmaligen Überschreiben von bereits in einem Playbook deklarierten Variablen.

In diesem Playbook ist ein Task definiert, der mit dem `file`-Modul ein Verzeichnis anlegt. Die Argumente werden als Variablen übergeben:

```
---
- name: a playbook with vars
  hosts: suse2
  vars:
    homedir: /home/tux/
```

## 15 Ansible Playbooks

```
  folder: foo
  filestate: directory
tasks:
  - name: a task to use the vars
    ansible.builtin.file:
      path: "{{ homedir }}/{{ folder }}"
      state: "{{ filestate }}
```

Wird das Playbook ausgeführt, so legt der Task das Verzeichnis /home/tux/foo an:

```
# ansible-playbook -i /root/code/inventory extravars.yml

PLAY [a playbook with vars] **

TASK [Gathering Facts] **
ok: [suse2]

TASK [a task to use the vars] **
changed: [suse2]

PLAY RECAP **
suse2 : ok=2     changed=1     unreachable=0     failed=0

root@suse2 ~]# ls /home/tux/
foo
```

Mit dem Parameter --extra-vars können diese Variablen für das Playbook nun überschrieben werden:

```
# ansible-playbook -i /root/code/inventory extravars.yml --extra-vars
  ↪ "homedir=/home/tix/ folder=bar.txt filestate=touch"

PLAY [a playbook with vars] **

TASK [Gathering Facts] **
ok: [suse2]

TASK [a task to use the vars] **
changed: [suse2]

PLAY RECAP **
suse2 : ok=2     changed=1     unreachable=0     failed=0
```

Mit dem Aufruf des Playbooks und der Option --extra-vars wurden die im Playbook deklarierten Variablen überschrieben, sodass nun die Datei /home/tix/bar.txt statt wie zuvor das Verzeichnis /home/tux/foo angelegt wurde:

```
root@suse2 ~]# ls /home/tix/
bar.txt
```

## Extra-Vars aus einer Datei

Anstatt die auf der Kommandozeile einzulesenden Variablen direkt dort zu formulieren, können Sie diese auch aus einer Datei laden lassen. Dazu müssen die Variablen in einer Datei deklariert werden, im Beispiel `/root/code/extravars_vars.yml`:

```
---
homedir: /home/tix/
folder: foo.txt
filestate: touch
```

Diese Datei kann nun mit dem Zeichen `@` an die Option `--extra-vars` übergeben werden:

```
# ansible-playbook -i /root/code/inventory extravars.yml --extra-vars
    ↪ @/root/code/extravars_vars.yml

PLAY [a playbook with vars] **

TASK [Gathering Facts] **
ok: [suse2]

TASK [a task to use the vars] **
changed: [suse2]

PLAY RECAP **
suse2 : ok=2     changed=1     unreachable=0     failed=0
```

Nun werden im Playbook die in der Datei `extravars_vars.yml` deklarierten Variablen verwendet und das Playbook legt die Datei `/home/tix/foo.txt` an:

```
root@suse2 ~]# ls /home/tix/
bar.txt  foo.txt
```

Neben YAML-Dateien können die Variablen auch in Dateien im `.json` Format gespeichert und mit `--extra-vars` eingelesen werden.



## 15.5 Aufgabenteil



# Aufgabenteil – Variablen auf der Kommandozeile

- ① Führen Sie einen Ad-Hoc Befehl für das System centos1 aus, in dem Sie mit dem Modul `ansible.builtin.service` sicherstellen, dass der Dienst `sshd` gestartet wurde. Deklarieren Sie dafür eine Variable mit dem Wert `sshd` und übergeben Sie diese an das Modul.
- ② Führen Sie das Playbook aus der Musterlösung der Aufgabe 1 – „Variablen im Playbook“ aus und überschreiben Sie dabei die darin deklarierten Variablen mit Variablen, die Sie auf der Kommandozeile deklarieren und übergeben.
- ③ Wiederholen Sie die letzte Aufgabe, nur dass Sie die Variablen diesmal in einer YAML-Datei deklarieren und diese an `ansible-playbook` übergeben.

1. Führen Sie einen Ad-Hoc Befehl für das System `centos1` aus, in dem Sie mit dem Modul `ansible.builtin.service` sicherstellen, dass der Dienst `sshd` gestartet wurde. Deklarieren Sie dafür eine Variable mit dem Wert `sshd` und übergeben Sie diese an das Modul.
2. Führen Sie das Playbook aus der Musterlösung der Aufgabe 1 „Variablen im Playbook“ von S. 324 aus und überschreiben Sie dabei die darin deklarierten Variablen mit Variablen, die Sie auf der Kommandozeile deklarieren und übergeben.

## 15 Ansible Playbooks

Das Playbook von S. 324:

```
---
- name: a playbook where a variable will be declared and debugged
  hosts: ubuntu1
  vars:
    name1: tux
    os: linux
    company: b1
  tasks:
    - name: a task with a debug module
      ansible.builtin.debug:
        msg: "My name is {{ name1 }}, I love {{ os }} and I work
              ↪ at {{ company }}."
```

3. Wiederholen Sie die letzte Aufgabe, nur dass Sie die Variablen diesmal in einer YAML-Datei deklarieren und diese an `ansible-playbook` übergeben.

## Musterlösung

- Führen Sie einen Ad-Hoc Befehl für das System `centos1` aus, in dem Sie mit dem Modul `ansible.builtin.service` sicherstellen, dass der Dienst `sshd` gestartet wurde. Deklarieren Sie dafür eine Variable mit dem Wert `sshd` und übergeben Sie diese an das Modul:

```
# ansible centos1 -i inventory -m ansible.builtin.service -a
  ↪ "name={{ servicename }} state=started" --extra-vars
  ↪ "servicename:sshd"
```

- Führen Sie das Playbook aus der Musterlösung der Aufgabe 1 „Variablen im Playbook“ von S. 324 aus und überschreiben Sie dabei die darin deklarierten Variablen mit Variablen, die Sie auf der Kommandozeile deklarieren und übergeben:

Führen Sie das Playbook mit `ansible-playbook` aus und deklarieren Sie mit `--extra-vars` die Variablen `name1`, `os` und `company`:

```
# ansible-playbook -i /root/code/inventory
  ↪ extravars_exercise2.yml --extra-vars "name1=Geeko os=Suse
  ↪ company='the office'"

PLAY [a playbook where a variable will be declared and debugged]
  ↪ **

TASK [Gathering Facts] **
ok: [ubuntul]

          Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de
TASK [a task with a debug module] **
ok: [ubuntul] => {
    "msg": "My name is Geeko, I love Suse and I work at the
          ↪ office."
}

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

Das `ansible.builtin.debug`-Modul gibt nun die auf der Kommandozeile deklarierten Variablen wieder.

- Wiederholen Sie die letzte Aufgabe, nur dass Sie die Variablen diesmal in einer YAML-Datei deklarieren und diese an `ansible-playbook` übergeben:

Erstellen Sie eine YAML-Datei, z. B. `variablen.yml`, und tragen Sie die Variablen dort ein:

```
---
name1: Geeko
os: Suse
company: 'the office'
```

Führen Sie anschließend das Playbook unter Angabe der Datei aus:

## 15 Ansible Playbooks

```
# ansible-playbook -i /root/code/inventory
    ↪ extravars_exercise2.yml --extra-vars @variablen.yml

PLAY [a playbook where a variable will be declared and debugged]
    ↪ **

TASK [Gathering Facts] **
ok: [ubuntul]

TASK [a task with a debug module] **
ok: [ubuntul] => {
    "msg": "My name is Geeko, I love Suse and I work at the
        ↪ office."
}

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

## 15.6 When



# Die when-Bedingung

- Ausführung des Task wird von Bedingung abhängig gemacht
- Stichwort: `when`
- mehrere Bedingungen möglich
- wird üblicherweise am Ende eines Task definiert

Mit der `when`-Bedingung ist es möglich, das Ausführen von Tasks innerhalb von Playbooks von Bedingungen abhängig zu machen. Als Bedingungen können z. B. Werte von Variablen, registrierten Variablen oder Ansible Facts gesetzt werden. Die Bedingungen können dabei mit einer Reihe von Konditionaloperatoren definiert und nach dem *wenn, dann*-Prinzip geprüft werden.

# Die when-Bedingung

## Task mit einer when-Bedingung

```
- name: ensure package is installed when true
  ansible.builtin.package:
    name: nmap
    state: present
  when: ansible_os_family == "Debian"
```

## Task mit zwei per and-Operator verknüpften when-Bedingungen

```
- name: make sure package is present when conditions are met
  ansible.builtin.package:
    name: nmap
    state: present
  when: ansible_os_family == "Debian" and ansible_distribution == "Ubuntu"
```

Die when-Kondition wird am Ende des von ihr abhängenden Task mit dem entsprechenden Operator definiert. Im folgenden Beispiel wird der Task nur dann ausgeführt, wenn der Ansible Fact `ansible_os_family` der definierten Bedingung entspricht. Die Variablen werden bei der Verwendung von `when` ohne Klammern referenziert:

```
---
- name: show an example of a when conditional
  hosts: all
  tasks:
    - name: make sure package is present when condition is met
      ansible.builtin.package:
        name: nmap
        state: present
      when: ansible_os_family == "Debian"
```

Es können auch mehrere Bedingungen mit einem `or`-Operator verknüpft werden:

```
---
- name: show an example of the or-operator in a when conditional
  hosts: all
  tasks:
    - name: make sure package is present when conditions are met
      ansible.builtin.package:
        name: nmap
```

```

    state: present
when: ansible_os_family == "Debian" or ansible_os_family ==
      → "RedHat"

```

Oder auch mit einem and-Operator:

```
---
- name: show an example of the and-operator in a when conditional
  hosts: all
  tasks:
    - name: make sure package is present when conditions are met
      ansible.builtin.package:
        name: nmap
        state: present
    when: ansible_os_family == "Debian" and ansible_distribution ==
          → "Ubuntu"
```

Die Bedingung kann auch negativ definiert werden:

```
---
- name: show an example of a negation in a when conditional
  hosts: all
  tasks:
    - name: make sure package is present when its not Debian
      ansible.builtin.package:
        name: nmap
        state: present
    when: ansible_os_family != "Debian"
```

Das Paket nmap wird nur auf den Systemen installiert, auf denen die Bedingungen erfüllt werden. Die Systeme, auf die diese Bedingungen nicht zutreffen, werden von dem Task übersprungen.

## Operatoren für when

### Mögliche Operatoren für when:

Operator	Funktion
<code>==</code>	beide Werte sind <i>gleich</i>
<code>or</code>	der eine <i>oder</i> der andere Wert trifft zu
<code>and</code>	eine <i>und</i> weitere Bedingungen treffen zu
<code>!=</code>	die Werte sind <i>ungleich</i>
<code>in</code>	eine Bedingung trifft zu, wenn ein bestimmter Wert enthalten ist
<code>not in</code>	eine Bedingung trifft zu, wenn ein bestimmter Wert <i>nicht</i> enthalten ist

### Mögliche Operatoren für when-Konditionen:

OPERATOR:	BEDEUTUNG:
<code>==</code>	beide Werte sind <i>gleich</i>
<code>or</code>	der eine <i>oder</i> der andere Wert trifft zu
<code>and</code>	eine <i>und</i> weitere Bedingungen treffen zu
<code>!=</code>	die Werte sind <i>ungleich</i>
<code>in</code>	eine Bedingung trifft zu, wenn ein bestimmter Wert enthalten ist
<code>not in</code>	eine Bedingung trifft zu, wenn ein bestimmter Wert <i>nicht</i> enthalten ist



## when: result is

Task mit einer "when: result is"-Bedingung

```
- name: a task with a when result is condition
  ansible.builtin.debug:
    msg: "Should be fine, the service is running."
  when: apache_result is succeeded
```

- apache\_result ist eine registrierte Variable

Mögliche Operatoren für "when: result is":

Zustand	Bedeutung
success succeeded	Task wurde erfolgreich ausgeführt.
fail failed	Ausführen des Task ist fehlgeschlagen.
skip skipped	Task wurde übersprungen.

Mit `when: result is` können Sie das Ausführen von Tasks auch vom Ausgang eines vorangegangenen Task abhängig machen. Dabei wird die Bedingung an die Zustände `failed`, `succeeded` oder `skipped` geknüpft. Das Resultat des Task muss dafür in einer Variable registriert werden, aus der `when` selbstständig den Zustand auslesen kann. Dabei können die Bedingungen mit den schon genannten Operatoren kombiniert werden:

```
---
- name: a playbook where when will trigger tasks depending on the
  ↪ results
  hosts: ubuntu1
  tasks:
    - name: the first task that will supply the result the other tasks
      ↪ are based on
      ansible.builtin.service:
        name: apache2
        state: started
        register: apache_result
        ignore_errors: True

    - name: this task is here to make sure that the package is present
      ↪ in the event, that the first task failed
      ansible.builtin.package:
        name: apache2
        state: present
```

## 15 Ansible Playbooks

```
register: apache_package
when: apache_result is failed

- name: this is just the announcement that the package present in
      ↪ the event, that the service failed
  ansible.builtin.debug:
    msg: "The service failed before so that this tasks makes sure
          ↪ it is there now."
  when: apache_package is not skipped

- name: this message appears, when the service had been started
      ↪ successfully
  ansible.builtin.debug:
    msg: "Should be fine, the service is running."
  when: apache_result is succeeded and apache_package is skipped
```

Die Anweisung „`ignore_errors: True`“ verhindert, dass das Playbook durch das Scheitern des Task an dieser Stelle abgebrochen wird. Das Resultat des Task wird in der Variable `apache_result` registriert. Enthält die Variable den Wert `failed`, wird der zweite Task ausgeführt, der das Paket `apache2` installiert. Das Resultat dieses Task wird wiederum in der Variable `apache_package` registriert. Enthält nun `apache_package` *nicht* den Wert `skipped`, wird mit dem `ansible.builtin.debug`-Modul in einem dritten Task eine Nachricht ausgegeben:

```
# ansible-playbook -i /root/code/inventory /root/code/when_is.yml

PLAY [a playbook where when will trigger tasks depending on the
      ↪ results] **

TASK [Gathering Facts] **
ok: [ubuntul]

TASK [the first task that will supply the result the other tasks are
      ↪ based on] **
fatal: [ubuntul]: FAILED! => {"changed": false, "msg": "Unable to
      ↪ start service apache2: Failed to start apache2.service: Unit
      ↪ apache2.service is masked.\n"}
...ignoring

TASK [this task is here to make sure that the package is present in
      ↪ the event, that the first task failed] **
changed: [ubuntul]

TASK [this is just the announcement that the package present in the
      ↪ event, that the service failed] **
ok: [ubuntul] => {
  "msg": "The service failed before so that we installed it now."
}

TASK [this message appears, when the service had been started
      ↪ successfully] **
skipping: [ubuntul]

PLAY RECAP **
```

```
ubuntu1 : ok=4      changed=1      unreachable=0      failed=0
```

Wird das Playbook hinterher ein zweites Mal aufgerufen, so verändern sich die Werte in den registrierten Variablen der Tasks. Dadurch wird ausschließlich der vierte Task ausgeführt, der lediglich eine Nachricht ausgibt:

```
# ansible-playbook -i /root/code/inventory /root/code/when_is.yml

PLAY [a playbook where when will trigger tasks depending on the
  ↪ results] **

TASK [Gathering Facts] **
ok: [ubuntu1]

TASK [the first task that will supply the result the other tasks are
  ↪ based on] **
changed: [ubuntu1]

TASK [this task is here to make sure that the package is present in
  ↪ the event, that the first task failed] **
skipping: [ubuntu1]

TASK [this is just the announcement that the package present in the
  ↪ event, that the service failed] **
skipping: [ubuntu1]

TASK [this message appears, when the service had been started
  ↪ successfully] **
ok: [ubuntu1] => {
    "msg": "Should be fine, the service is running."
}

PLAY RECAP **
ubuntu1 : ok=3      changed=1      unreachable=0      failed=0
```

ZUSTAND:	BEDEUTUNG:
success succeeded	Task wurde erfolgreich ausgeführt.
fail failed	Ausführen des Task ist fehlgeschlagen.
skip skipped	Task wurde übersprungen.

## changed\_when & failed\_when

- selber festlegen, wann ein Task als erfolgreich (*changed*) oder gescheitert (*failed*) gilt

### Task mit einer changed\_when-Bedingung:

```
- name: first task
  ansible.builtin.command: "httpd -k start"
  register: httpd_register
  changed_when: "'string' not in httpd_register['stdout']"
```

### Task mit einer failed\_when-Bedingung:

```
- name: a task that starts a service via the command module
  ansible.builtin.command: "httpd -k start"
  register: httpd_register
  failed_when: "'already running' not in httpd_register['stdout'] \
    and httpd_register['rc'] !=0"
```

Nutzen Sie mit Ansible die Module `ansible.builtin.command` oder `ansible.builtin.shell`, um auf einem Zielsystem direkt Befehle oder Skripte auszuführen, so kann Ansible zwar erkennen, ob das Modul erfolgreich ausgeführt wurde, jedoch nicht, ob der Befehl oder das Skript selbst erfolgreich ausgeführt wurden.

In diesem Fall lassen sich mit `changed_when` und `failed_when` Bedingungen definieren, unter welchen Umständen der Prozess erfolgreich beendet wurde (*changed*) und unter welchen Umständen der Prozess gescheitert ist (*failed*).

Im folgenden Beispiel wird mit dem `ansible.builtin.command`-Modul sichergestellt, dass der Dienst `httpd` gestartet wurde:

```
---
- name: a playbook with changed_when
  hosts: centos2
  tasks:
    - name: a task that starts a service via the command module
      ansible.builtin.command: "httpd -k start"
```

Bei Ausführung des Playbook erhalten Sie die Ausgabe von Ansible, dass der Task erfolgreich ausgeführt (*changed*) wurde:

```
# ansible-playbook -i /root/code/inventory
  ↪ /root/code/changed_when_httpd.yml
```

```

PLAY [a playbook with some changed_when] **

TASK [Gathering Facts] **
ok: [centos2]

TASK [a task that starts a service via the command module] **
changed: [centos2]

PLAY RECAP **
centos2 : ok=2     changed=1     unreachable=0     failed=0

```

Diese Ausgabe ändert sich jedoch nicht, wenn der Dienst bereits läuft. In diesem Fall würde die Ausgabe `stdout` des Befehls selbst zeigen, dass der Befehl in dem Modul nicht erneut ausgeführt wird, wenn der Dienst bereits läuft:

```

TASK [task with debug module to show the output from stdout of "httpd
    ↪ -k start"] **
ok: [centos2] => {
    "msg": "httpd (pid 2033) already running"
}

```

Ansible würde in diesem Fall den Task trotzdem als erfolgreich ausgeführt (*changed*) ausgeben, weil das Modul erfolgreich ausgeführt wurde. Da für den Befehl selbst diese Ausgabe nur dann zutrifft, wenn der Dienst `httpd` auch gestartet wurde und nicht schon lief, lässt sich mit `changed_when` eine Bedingung definieren, unter der Ansible den Task als *changed* ausgibt:

```

---
- name: a playbook with changed_when
  hosts: centos2
  tasks:

    - name: first task
      ansible.builtin.command: "httpd -k start"
      register: httpd_register
      changed_when: "'already running' not in httpd_register['stdout']"

```

Zunächst muss dafür die Ausgabe des Befehls in einer Variable registriert werden. Im Anschluss daran kann mit `changed_when` die Bedingung definiert werden, unter welchen Umständen der Task als erfolgreich ausgeführt gelten soll. In diesem Fall wäre dies, wenn sich der String `already running` nicht in der Standardausgabe `stdout` befindet:

```

# ansible-playbook -i /root/code/inventory
    ↪ /root/code/changed_when_httpd.yml

PLAY [a playbook with some changed_when] **

TASK [Gathering Facts] **
ok: [centos2]

TASK [a task that starts a service via the command module] **
ok: [centos2]

```

## 15 Ansible Playbooks

```
PLAY RECAP **  
centos2 : ok=2     changed=0     unreachable=0     failed=0
```

Somit wird der Befehl in dem Task ausgeführt, der Zustand jedoch nicht als geändert markiert, da der Dienst bereits lief.

Ähnlich verhält es sich, wenn der Befehl oder das Skript, das gestartet wird, nicht erfolgreich ausgeführt wird. Auch in diesem Fall würde Ansible ausgeben, dass das Modul selbst erfolgreich ausgeführt wurde. Für diesen Fall lassen sich analog zu `changed_when` mit `failed_when` Bedingungen definieren, unter denen der Prozess als gescheitert (*failed*) gilt:

```
---  
- name: a playbook with failed_when  
  hosts: centos2  
  tasks:  
  
    - name: a task that starts a service via the command module  
      ansible.builtin.command: "httpd -k start"  
      register: httpd_register  
      failed_when: "'already running' not in httpd_register['stdout']  
                   and httpd_register['rc'] !=0"
```

In diesem Fall wird der Task als gescheitert ausgegeben, wenn der String `already running` nicht in der Standardausgabe enthalten ist und der Return Code etwas anderes als den Wert „0“ enthält.

Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de



## when: in|not in

- Bedingung davon abhängig machen, ob ein String in einer (registrierten) Variable enthalten ist

### Task mit einer "when: in"-Bedingung:

```
- name: a task where a result is registered in a variable
  ansible.builtin.command: file /home/tux/test
  register: tuxfile_result

- name: show a message if string is in the registered var
  ansible.builtin.debug:
    msg: "Show this if the command 'file' couldn't find it"
    when: "'cannot' in tuxfile_result['stdout']"
```

Sie können auch bestimmte Ausgaben von Befehlen zu Bedingungen von `when`-Konditionen machen. Dies bietet sich z. B. ebenfalls dann an, wenn in einem Task kein Ansible-Modul, sondern ein Befehl direkt ausgeführt wird, da Ansible den Ausgang dieser Befehle nicht überprüft. Mit `when: 'example'` in `$variable` lässt sich eine Bedingung definieren, bei der ein bestimmter Wert in einer Variable gegeben sein muss:

```
---
- name: a playbook with a when condition
  hosts: centos1
  tasks:
    - name: a task where a result will be registered in a variable
      ansible.builtin.command: file /home/tux/test
      register: tuxfile_result

    - name: show a message if string is in the registered var
      ansible.builtin.debug:
        msg: "Show this Message if the command 'file' could not find
              ↪ the file"
        when: "'cannot' in tuxfile_result['stdout']"
```

Ist der Wert „cannot“ im Dictionary-Eintrag `stdout` der registrierten Variable `tuxfile_result` enthalten, wird der Task ausgeführt, der in diesem Fall mit dem `debug`-Modul eine Nachricht ausgibt:

## 15 Ansible Playbooks

```
# ansible-playbook -i inventory when_in.yml

PLAY [a playbook with a when condition] **

TASK [Gathering Facts] **
ok: [centos1]

TASK [a task where a result will be registered in a variable] **
changed: [centos1]

TASK [debug] **
ok: [centos1] => {
    "msg": "Show this Message if the command 'file' could not find the
        ↪ file"
}

PLAY RECAP **
centos1 : ok=3      changed=1      unreachable=0      failed=0
```

Zusätzlich zu der Bedingung, dass ein Wert in der Variable enthalten sein muss, kann die Bedingung auch so gestellt werden, dass ein Wert *nicht* in der Variable enthalten sein soll. Mit dem Ausdruck `when: 'example' not in $variable` kann der Wert festgelegt werden:

```
---
- name: a playbook with a when condition
  hosts: centos1
  tasks:

    - name: a task where a result will be registered in a variable
      ansible.builtin.command: file /home/tux/test
      register: tuxfile_result

    - name: a debug task
      ansible.builtin.debug:
        msg: "Show this message if when there seems to be a file"
        when: "'cannot' not in tuxfile_result['stdout']"
```

Nun wird der Task immer dann ausgeführt, wenn der Wert nicht in der Ausgabe der registrierten Variable enthalten ist:

```
# ansible-playbook -i /root/code/inventory /root/code/when_in.yml

PLAY [a playbook with a when condition] **

TASK [Gathering Facts] **
ok: [centos1]

TASK [a task where a result will be registered in a variable] **
changed: [centos1]

TASK [a debug task] **
ok: [centos1] => {
    "msg": "Show this Message if when there seems to be a file"
```

```
}
```

```
PLAY RECAP **  
centos1 : ok=3      changed=1      unreachable=0      failed=0
```



### 15.6.1 Aufgabenteil



## Aufgabenteil – when 1/2

- ① Schreiben Sie ein Playbook mit einem Task, in dem mit dem Modul `ansible.builtin.package` sichergestellt wird, dass das Programm `git` installiert ist, sofern die Zielsysteme den `ansible_hostname` `ubuntu1` oder `suse2` haben.
- ② Fügen Sie dem Playbook einen weiteren Task hinzu, in dem mit dem Modul `ansible.builtin.package` sichergestellt wird, dass das Paket `curl` auf allen Systemen vorhanden ist, die *nicht* zur `ansible_os_family` Suse und Debian gehören.

1. Schreiben Sie ein Playbook mit einem Task, in dem mit dem Modul `ansible.builtin.package` sichergestellt wird, dass das Programm `git` installiert ist, sofern die Zielsysteme den `ansible_hostname` `ubuntu1` oder `suse2` haben.
2. Fügen Sie dem Playbook einen weiteren Task hinzu, in dem mit dem Modul `ansible.builtin.package` sichergestellt wird, dass das Paket `curl` auf allen Systemen vorhanden ist, die *nicht* zur `ansible_os_family` Suse und Debian gehören.

## Musterlösung

- Schreiben Sie ein Playbook mit einem Task, in dem mit dem Modul `ansible.builtin.package` sichergestellt wird, dass das Programm `git` installiert ist, sofern die Zielsysteme den `ansible_hostname` `ubuntul` oder `suse2` haben:

In dem Playbook muss mit `when` definiert sein, dass der Task nur auf einem System ausgeführt wird, bei dem der Fact `ansible_hostname` gleich `ubuntul` oder `suse2` ist:

```
---
- name: make sure the right software is present when the
  ↪ conditions are met
hosts: all
tasks:
  - name: make sure the package git is present when the system
    ↪ has the hostname ubuntul or suse2
    ansible.builtin.package:
      name: git
      state: present
    when: ansible_hostname == "ubuntul" or ansible_hostname ==
          ↪ "suse2"
```

Ausführung des Playbooks:

```
# ansible-playbook -i /root/code/inventory when_first_exercise.yml

PLAY [all] **

TASK [Gathering Facts] **
ok: [ubuntul]
ok: [ubuntu2]
ok: [suse1]
ok: [suse2]
ok: [centos1]
ok: [centos2]

TASK [make sure the package git is present when the system has
  ↪ the hostname ubuntul or suse2] **
skipping: [suse2]
skipping: [centos1]
skipping: [centos2]
skipping: [ubuntu2]
changed: [ubuntul]
changed: [suse2]

PLAY RECAP **
centos1      : ok=1    changed=0    unreachable=0    failed=0
centos2      : ok=1    changed=0    unreachable=0    failed=0
suse1        : ok=1    changed=0    unreachable=0    failed=0
suse2        : ok=2    changed=1    unreachable=0    failed=0
ubuntul      : ok=2    changed=1    unreachable=0    failed=0
ubuntu2      : ok=1    changed=0    unreachable=0    failed=0
```

- 2. Fügen Sie dem Playbook einen weiteren Task hinzu, in dem mit dem Modul `ansible.builtin.package` sichergestellt wird, dass das Paket `curl` auf allen Systemen vorhanden ist, die *nicht* zur `ansible_os_family` Suse und Debian gehören:**

Im Playbook muss mit `when` für den zweiten Task definiert sein, dass der Task nur auf Systemen ausgeführt wird, die *nicht* zur `ansible_os_family` Suse oder Debian gehören:

```
---
- name: make sure the right software is present when the
  ↪ conditions are met
hosts: all
tasks:
  - name: make sure the package git is present when the system
    ↪ has the hostname ubuntul or suse2
    ansible.builtin.package:
      name: git
      state: present
    when: ansible_hostname == "ubuntul" or ansible_hostname
      ↪ == "suse2"

  - name: make sure the package curl is present when the system
    ↪ does not belong to the os family Debian and Suse
    ansible.builtin.package:
      name: curl
      state: present
    when: ansible_os_family != "Suse" and ansible_os_family !=
      ↪ "Debian"
```

Ausführung des Playbooks:

```
# ansible-playbook -i /root/code/inventory
  ↪ when_second_exercise.yml

PLAY [all] **

TASK [Gathering Facts] **
ok: [ubuntul]
ok: [ubuntu2]
ok: [suse1]
ok: [suse2]
ok: [centos2]
ok: [centos1]

TASK [make sure the package git is present when the system has
  ↪ the hostname ubuntul or suse2] **
skipping: [suse1]
skipping: [centos1]
skipping: [centos2]
skipping: [ubuntu2]
ok: [suse2]
ok: [ubuntul]
```

## 15 Ansible Playbooks

```
TASK [make sure the package curl is present when the system does
    → not belong to the os family Debian and Suse] **
skipping: [suse1]
skipping: [suse2]
skipping: [ubuntu1]
skipping: [ubuntu2]
ok: [centos2]
ok: [centos1]

PLAY RECAP ***
centos1      : ok=2      changed=0      unreachable=0      failed=0
centos2      : ok=2      changed=0      unreachable=0      failed=0
suse1        : ok=1      changed=0      unreachable=0      failed=0
suse2        : ok=2      changed=0      unreachable=0      failed=0
ubuntu1     : ok=2      changed=0      unreachable=0      failed=0
ubuntu2     : ok=1      changed=0      unreachable=0      failed=0
```



## Aufgabenteil – when 2/2

- ③ Schreiben Sie ein Playbook, in dem auf bestimmten Systemen der Webserver apache2 installiert wird. Deklarieren Sie für die distributionsspezifischen Pakete jeweils eine Variable. Definieren Sie when-Bedingungen, nach denen das jeweils richtige Paket installiert wird. Schreiben Sie einen weiteren Task, in dem der jeweils richtige Dienst gestartet wird.
- ④ Schreiben Sie ein Playbook, in dem das Resultat eines Tasks in einer Variable registriert wird. Erstellen Sie zwei weitere Tasks, in denen mit when-Konditionen Tasks in Abhängigkeit zum Wert der registrierten Variable ausgeführt werden.

3. Schreiben Sie ein Playbook, in dem mit dem Modul `ansible.builtin.package` sichergestellt wird, dass auf den Systemen `suse1` und `centos1` der Webserver `apache2` installiert ist. Deklarieren sie für das jeweils distributionsabhängige Installationspaket (`apache2` oder `httpd`) eine Variable. Machen Sie die Wahl des Pakets für das jeweilige System von einer Bedingung abhängig, die mit `when` abgefragt wird. Schreiben Sie einen weiteren Task, der mit dem Modul `ansible.builtin.service` sicherstellt, dass der Webserverdienst auf dem jeweiligen System gestartet wurde. Erweitern Sie auch hierfür Ihren Task um die Abhängigkeit von einer `when`-Kondition.
4. Schreiben Sie ein Playbook, in dem Sie in einem Task mit dem Modul `ansible.builtin.stat` prüfen, ob die Datei `/root/example/stat.txt` auf dem System `suse2` vorhanden ist und registrieren Sie das Ergebnis in einer Variable. Fügen Sie zwei weitere Tasks hinzu. Ein Task soll, sofern die `when`-Bedingung nicht zutrifft und die Datei `stat.txt` nicht vorhanden ist, mit dem `ansible.builtin.debug`-Modul eine Nachricht ausgeben, die dies erklärt. Ein weiterer Task soll bei Erfüllung der `when`-Bedingung mit dem `ansible.builtin.file`-Modul einen symbolischen Link nach `/root/stat.txt` erzeugen.

## Musterlösung

3. Schreiben Sie ein Playbook, in dem mit dem Modul `ansible.builtin.package` sichergestellt wird, dass auf den Systemen `suse1` und `centos1` der Webserver `apache2` installiert ist. Deklarieren Sie für das jeweils distributionsabhängige Installationspaket (`apache2` oder `httpd`) eine Variable. Machen Sie die Wahl des Pakets für das jeweilige System von einer Bedingung abhängig, die mit `when` abgefragt wird. Schreiben Sie einen weiteren Task, der mit dem Modul `ansible.builtin.service` sicherstellt, dass der Webserverdienst auf dem jeweiligen System gestartet wurde. Erweitern Sie auch hierfür Ihren Task um die Abhängigkeit von einer `when`-Kondition:

Ein Playbook, das dieser Aufgabe entspricht, könnte wie folgt aussehen:

```
---
- name: A playbook to make sure a webserver is installed on
  # certain systems
  hosts: suse1, centos1
  vars:
    webserver_suse: apache2
    webserver_centos: httpd
  tasks:

    - name: make sure the webserver package is present on the
      # suse system
      ansible.builtin.package:
        name: "{{ webserver_suse }}"
        state: present
      when: ansible_distribution_file_variety == "SUSE"

    - name: make sure the webserver package is present on the
      # centos system
      ansible.builtin.package:
        name: "{{ webserver_centos }}"
        state: present
      when: ansible_distribution == "CentOS"

    - name: make sure the service is running on the suse system
      ansible.builtin.service:
        name: "{{ webserver_suse }}"
        state: started
      when: ansible_distribution_file_variety == "SUSE"

    - name: make sure the service is running on the centos system
      ansible.builtin.service:
        name: "{{ webserver_centos }}"
        state: started
      when: ansible_distribution == "CentOS"
```

In diesem Fall prüft die `when`-Kondition, ob es sich bei dem System entweder um ein SUSE oder um ein CentOS handelt. Diese Überprüfung findet hier anhand von `ansible_facts` statt. Es stehen grundsätzlich verschiedene Möglichkeiten für diese Überprüfung zu Verfügung, so wird hier in einem Fall nach dem Fact

`ansible_distribution_file_variety` gefragt und im anderen Fall nach der `ansible_distribution`. Es wären an dieser Stelle auch andere Bedingungen wie z.B. `ansible_hostname` oder dergleichen möglich. Anhand der Erfüllung oder der Nichterfüllung der jeweiligen Bedingung wird dann der entsprechende Task ausgeführt, in dem das für die Distribution bestimmte Paket aus einer Variable ausgelesen wird.

Ausführung des Playbooks:

```
# ansible-playbook -i /root/code/inventory
  ↪ /root/code/when_third_exercise.yml

PLAY [A playbook to install a webserver on certain systems] **

TASK [Gathering Facts] **
ok: [suse1]
ok: [centos1]

TASK [make sure the webserver package is present on the suse
  ↪ system] **
skipping: [centos1]
changed: [suse1]

TASK [make sure the webserver package is present on the centos
  ↪ system] **
skipping: [suse1]
changed: [centos1]

TASK [make sure the service is running on the suse system] **
skipping: [centos1]Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de
changed: [suse1]

TASK [make sure the service is running on the centos system] **
skipping: [suse1]
changed: [centos1]

PLAY RECAP **
centos1      : ok=3    changed=2    unreachable=0    failed=0
suse1       : ok=3    changed=2    unreachable=0    failed=0
```

4. Schreiben Sie ein Playbook, in dem Sie in einem Task mit dem Modul `ansible.builtin.stat` prüfen, ob die Datei `/root/example/stat.txt` auf dem System `suse2` vorhanden ist und registrieren Sie das Ergebnis in einer Variable. Fügen Sie zwei weitere Tasks hinzu. Ein Task soll, sofern die `when`-Bedingung nicht zutrifft und die Datei `stat.txt` nicht vorhanden ist, mit dem `ansible.builtin.debug`-Modul eine Nachricht ausgeben, die dies erklärt. Ein weiterer Task soll bei Erfüllung der `when`-Bedingung mit dem `ansible.builtin.file`-Modul einen symbolischen Link nach `/root/stat.txt` erzeugen:

Beispiel für ein Playbook:

```
---
- name: make sure a certain file is at its place, using the when
  ↪ conditional
  hosts: suse2
  tasks:
    - name: check if the file is there and register the result to
      ↪ a variable
      ansible.builtin.stat:
        path: /root/example/stat.txt
        register: stat_result

    - name: show a message if condition untrue
      ansible.builtin.debug:
        msg: "the file stat.txt is not present"
        when: stat_result['stat']['exists'] == false

    - name: make sure link is set if condition true
      ansible.builtin.file:
        src: /root/example/stat.txt
        dest: /root/stat.txt
        state: link
        when: stat_result['stat']['exists'] == true
```

Im ersten Task wird mit dem Modul `stat` geprüft, ob die Datei vorhanden ist. Die `when`-Kondition fragt, ob die Bedingung, dass der Dictionary-Eintrag aus der registrierten Variable `stat_result` dem Resultat „false“ entspricht, wahr ist. Trifft diese Bedingung zu, wird im zweiten Task mit dem Modul `ansible.builtin.debug` und dessen Parameter `msg` eine Nachricht ausgegeben:

```
# ansible-playbook -i inventory when_fourth_exercise.yml
```

```
PLAY [make sure a certain file is at its place, using the when
  ↪ conditional] *

TASK [Gathering Facts] **
ok: [suse2]

TASK [check if the file is there and register the result to a
  ↪ variable] *
ok: [suse2]
```

```

TASK [show a message if condition untrue] *
ok: [suse2] => {
    "msg": "the file stat.txt is not present"
}

TASK [make sure link is set if condition true] *
skipping: [suse2]

PLAY RECAP **
suse2 : ok=3    changed=0    unreachable=0
      failed=0   skipped=1   rescued=0   ignored=0

```

Der dritte Task wird ausgeführt, wenn die Datei vorhanden ist und die Prüfung des `when`-Konditionals wahr ist. In diesem Fall wird mit dem `ansible.builtin.file`-Modul und dem `state: touch` sichergestellt, dass ein symbolischer Link von der Datei `stat.txt` nach `/root/` existiert:

```

# ansible-playbook -i inventory when_fourth_exercise.yml

PLAY [make sure a certain file is at its place, using the when
      ↪ conditional] **

TASK [Gathering Facts] **
ok: [suse2]

TASK [check if the file is there and register the result to a
      ↪ variable] **
ok: [suse2]

TASK [show a message if condition untrue] **
skipping: [suse2]

TASK [make sure link is set if condition true] **
changed: [suse2]

PLAY RECAP **
suse2 : ok=3    changed=1    unreachable=0
      failed=0   skipped=1   rescued=0   ignored=0

```

Der Task für den die Bedingung nicht zutrifft wird jeweils von Ansible übersprungen und mit einem `skipped` versehen.

## 15.7 loop



# Die loop-Schleife

- Standardschleife
- genutzt für einfache Wiederholungen
- iteriert über eine Liste von Werten oder Objekten
- Beispiel:
  - Liste von Benutzern
  - Liste von Tasks
  - Liste von Files
- hieß bis ansible 2.4 `with_items`

Mit `loop` stellt Ansible eine Möglichkeit zu Verfügung, Schleifen in Playbooks einzubauen. Dies hat den Vorteil, dass einzelne Tasks wiederholt ausgeführt und dabei Listen für bestimmte Aufgaben iteriert werden können. Dabei gibt es eine Vielzahl an Möglichkeiten, Listen zu verwenden, oder `loop` mit anderen komplexen Werkzeugen wie z. B. Variablen oder dem `when`-Konditional zu kombinieren. Die `loop`-Schleife wird am Ende eines Task gesetzt und kann ggf. mit Optionen versehen werden.



### Hinweis Übergang von `with_*` zu `loop`

Seit Ansible Version 2.5 ersetzt `loop` das vorher verwendete `with_*` in fast allen Anwendungsbereichen. Es ist geplant, in zukünftigen Versionen nur noch `loop` einzusetzen. Daher bezieht sich auch diese Unterlage maßgeblich auf `loop` und nicht auf das als veraltet zu betrachtende `with_*`. Die `with_*`-Syntax bleibt allerdings für die absehbare Zukunft weiterhin gültig.



## loop mit einer Liste

- `item` ist die reservierte `loop`-Variable
- wird als Iterationsobjekt verwendet

Sicherstellen, dass mehrere Nutzer vorhanden sind:

```
- name: make sure users are present
  ansible.builtin.user:
    name: "{{ item }}"
    state: present
  loop:
    - 'user1'
    - 'user2'
```

Die `loop`-Schleife wird zum Iterieren von Listen genutzt um so eine Aufgabe mit verschiedenen Punkten aus dieser Liste zu wiederholen.

Eine einfache Liste kann direkt nach dem Aufruf von `loop` erstellt werden:

```
loop:
  - 'user1'
  - 'user2'
```

Die Liste wird als `item` in geschweiften Klammern referenziert, d.h. in der gleichen Schreibweise wie auch Variablen:

```
- name: make sure users are present
  ansible.builtin.user:
    name: "{{ item }}"
    state: present
```

Die Schleife mitsamt der Liste wird einfach dem Task angehängt, in dem das `item` referenziert werden soll. Das `item` bezeichnet dabei das Iterationsobjekt von `loop`. In diesem Fall wird mit der Schleife sichergestellt, dass alle Benutzer der Liste auf dem Zielsystem vorhanden sind:

```
---
- name: a playbook with a simple list in a loop
  hosts: centos1
```

## 15 Ansible Playbooks

```
tasks:  
  
  - name: make sure users are present  
    ansible.builtin.user:  
      name: "{{ item }}"  
      state: present  
    loop:  
      - 'user1'  
      - 'user2'
```

Das `ansible.builtin.user`-Modul iteriert nun durch die mit `loop` definierte Liste und wiederholt den Task für jeden in der Liste aufgezählten Punkt:

```
# ansible-playbook -i /root/code/inventory loop.yml  
  
PLAY [a playbook with a simple list in a loop] **  
  
TASK [Gathering Facts] **  
ok: [centos1]  
  
TASK [add users] **  
changed: [centos1] => (item=user1)  
changed: [centos1] => (item=user2)  
  
PLAY RECAP **  
centos1 : ok=2     changed=1     unreachable=0     failed=0
```

In einem Playbook ohne Schleife wären hierfür zwei Tasks nötig:

```
---  
- name: a playbook with two almost identical tasks instead of a simple  
  ↪ list in a loop  
  hosts: centos1  
  tasks:  
  
  - name: make sure user1 is there  
    ansible.builtin.user:  
      name: user1  
      state: present  
  
  - name: make sure user2 is there  
    ansible.builtin.user:  
      name: user2  
      state: present
```



# loop über eine Liste aus Variablen

- loop wird die Variable übergeben

Sicherstellen, dass mehrere Pakete installiert sind:

```
---
- hosts: centos1
  vars:
    directories:
      - /home/tux/foo
      - /opt/bar
      - /tmp/work
  tasks:
    - name: makes sure the dirs are present
      ansible.builtin.file:
        path: "{{ item }}"
        state: directory
      loop: "{{ directories }}"
```

Wenn verschiedene Tasks oder Plays mit loop dieselbe Liste abarbeiten sollen, so kann die Liste auch als Variable für das Playbook deklariert werden.

```
---
- name: A Playbook, that uses loops and a list in a Variable to make
  ↪ sure that certain directories are present
  hosts: centos1
  vars:
    directories:
      - /home/tux/foo
      - /opt/bar
      - /tmp/work
  tasks:
    - name: makes sure the dirs are present
      ansible.builtin.file:
        path: "{{ item }}"
        state: directory
      loop: "{{ directories }}"
```

## 15 Ansible Playbooks

Bei Ausführung des Playbooks:

```
# ansible-playbook -i inventory directories.yml

PLAY [A Playbook, that uses loops and a list in a Variable to make
      ↳ sure that certain directories are present] **

TASK [Gathering Facts] **
ok: [centos1]

TASK [makes sure the dirs are present] **
changed: [centos1] => (item=/home/tux/foo)
changed: [centos1] => (item=/opt/bar)
changed: [centos1] => (item=/tmp/work)

PLAY RECAP **
centos1 : ok=2    changed=1    unreachable=0
      ↳ failed=0    skipped=0    rescued=0    ignored=0
```

Das Modul `ansible.builtin.file` arbeitet die als Variable deklarierte Liste `directories` mit `loop` ab und stellt sicher, dass die Verzeichnisse `/home/tux/foo`, `/opt/bar` und `/tmp/work` auf dem Zielhost `centos1` vorhanden sind.

Die Liste als Variable zu deklarieren hat den Vorteil, dass sie so auch von anderen Tasks benutzt werden kann:

```
---
- name: A Playbook, that uses loops and a list in a Variable to make
      ↳ sure that certain directories are present
hosts: centos1
vars:
  directories:
    - /home/tux/foo
    - /opt/bar
    - /tmp/work
tasks:
  - name: makes sure the dirs are present
    ansible.builtin.file:
      path: "{{ item }}"
      state: directory
    loop: "{{ directories }}"
  - name: make sure there is a readme in the dirs
    ansible.builtin.file:
      dest: "{{ item }}/readme.md"
      state: touch
    loop: "{{ directories }}"
```

Hier bedient sich ein zweiter Task mit `loop` der deklarierten Liste und stellt sicher, dass sich in den Verzeichnissen eine Datei `readme.md` befindet:

```
# ansible-playbook -i inventory directories.yml

PLAY [A Playbook that uses loops and a list in a variable to make sure
      ↳ that certain directories are present] *
```

```
TASK [Gathering Facts] *
ok: [centos1]

TASK [makes sure the dirs are present] *
ok: [centos1] => (item=/home/tux/foo)
ok: [centos1] => (item=/opt/bar)
ok: [centos1] => (item=/tmp/work)

TASK [make sure there is a readme in the dirs] *
changed: [centos1] => (item=/home/tux/foo)
changed: [centos1] => (item=/opt/bar)
changed: [centos1] => (item=/tmp/work)

PLAY RECAP *
centos1 : ok=3    changed=1    unreachable=0
          ↢ failed=0    skipped=0    rescued=0    ignored=0
```

## Installation einer Liste von Paketen

- *nicht* mit loop
- Liste direkt dem Modul übergeben
- auch per Variable

### Übergabe einer Liste an das package-Modul

```
- name: make sure packages are present
  ansible.builtin.package:
    name:
      - 'nano'
      - 'curl'
      - 'tree'
    state: present
```

Zur Installation mehrerer Pakete auf einmal auf einem Zielsystem wird anstatt der Verwendung von `loop` empfohlen dem jeweiligen Modul direkt eine Liste der Pakete zu übergeben. Dies kann sowohl als Liste direkt am Modul als auch per Variable geschehen:

```
- name: make sure packages are present
  ansible.builtin.package:
    name:
      - 'nano'
      - 'curl'
      - 'tree'
    state: present
```

oder:

```
- name: make sure packages are present
  ansible.builtin.package:
    name: "{{ packagelist }}"
    state: present
```

Der Vorteil an diesem Vorgehen besteht darin dass das Paketmodul auf diese Art nur einmal aufgerufen wird und dabei mehrere Pakete installiert. Bei Übergabe der Liste der Pakete per `loop` an das Modul würde `loop` den Task und somit das Modul für jedes Listenobjekt erneut iterieren und ausführen und somit mehr Last auf dem Zielsystem erzeugen.



# loop mit einer Liste mit mehreren Werten

- loop eine Liste von Dictionaries ([]) übergeben
- über die in den ['keys'] enthaltenen Werte iterieren

## Eine Liste mit mehreren Key-Value-Paaren abarbeiten

```
- name: make sure users are present with groups
  ansible.builtin.user:
    name: "{{ item['name'] }}"
    group: "{{ item['group'] }}"
    state: present
  loop:
    - { name: 'user1', group: 'group1' }
    - { name: 'user2', group: 'group2' }
```

Mit `loop` können auch Listen iteriert und referenziert werden, die mehrere *dictionaries* mit Key-Value-Paaren enthalten. Die einzelnen Werte (*Values*) werden dabei über die in eckige Klammern gefassten *Keys* referenziert:

```
---
- name: Ensure that users are there
  hosts: suse2
  tasks:
    - name: make sure users from list are present
      ansible.builtin.user:
        name: "{{ item['name'] }}"
        uid: "{{ item['uid'] }}"
        shell: "{{ item['shell'] }}"
        state: present
      loop:
        - { name: 'tex', uid: '1350', shell: '/bin/bash' }
        - { name: 'tix', uid: '1751', shell: '/bin/bash' }
        - { name: 'tox', uid: '1054', shell: '/bin/bash' }
```

Hier werden die Benutzer, ihre `uid` und ihre `shell` als Werte aus der Liste unter `loop` entnommen und an den passenden Stellen in den Parametern des `user`-Moduls in eckigen Klammern referenziert. Das Ergebnis ist, dass der Liste entsprechend drei Benutzer mit jeweils individuellen Werten auf dem Zielsystem vorhanden sind:

## 15 Ansible Playbooks

```
# ansible-playbook -i /root/code/inventory loop1.yml

PLAY [Ensure that users are there] **

TASK [Gathering Facts] **
ok: [suse2]

TASK [make sure users from list are present] **
changed: [suse2] => (item={u'shell': u'/bin/bash', u'name': u'tex',
    ↪ u'uid': u'1350'})
changed: [suse2] => (item={u'shell': u'/bin/bash', u'name': u'tix',
    ↪ u'uid': u'1751'})
changed: [suse2] => (item={u'shell': u'/bin/bash', u'name': u'tox',
    ↪ u'uid': u'1054'})

PLAY RECAP **
suse2 : ok=2     changed=1     unreachable=0     failed=0
```

Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de



## loop über ein Dictionary

- über sowohl *Key* als auch *Value* iterieren
- Jinja2 Filter `dict2items`

### Über Key und Value aus Dictionaries iterieren

```
- hosts: ubuntu2
  vars:
    fruits:
      apple: 'Red'
      orange: 'Orange'
      grapes: 'Green'
  tasks:
    - name: loop through the dictionary Fruits
      ansible.builtin.debug:
        msg: "key={{ item['key'] }} val={{ item['value'] }}"
      loop: "{{ fruits|dict2items }}"
```

Mit `loop` lassen sich auch die „Key:Value“-Paare aus Dictionaries referenzieren und iterieren. Dafür müssen die Einträge aus dem Dictionary für `loop` mit dem Jinja2-Filter `dict2items` in `items` umgewandelt werden:

```
- hosts: ubuntu2
  vars:
    fruits:
      apple: 'Red'
      orange: 'Orange'
      grapes: 'Green'
  tasks:
    - name: a loop through a dictionary
      ansible.builtin.debug:
        msg: " key is {{ item['key'] }} and val is {{ item['value'] }}"
      loop: "{{ fruits|dict2items }}"
```

Die einzelnen Werte werden in eckigen Klammern entweder als `key` oder als `value` referenziert. Somit sind sie für `loop` auswertbar:

```
# ansible-playbook -i /root/code/inventory loop_dict.yml

PLAY [dict with a loop] *

TASK [Gathering Facts] **
```

## 15 Ansible Playbooks

```
ok: [ubuntul]

TASK [a loop through a dictionary] **
ok: [ubuntul] => (item={'key': u'Orange', 'value': u'Orange'}) => {
    "msg": " key is Orange and val is Orange"
}
ok: [ubuntul] => (item={'key': u'Apple', 'value': u'Red'}) => {
    "msg": " key is Apple and val is Red"
}
ok: [ubuntul] => (item={'key': u'Grapes', 'value': u'Green'}) => {
    "msg": " key is Grapes and val is Green"
}

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

### 15.7.1 Aufgabenteil



## Aufgabenteil – loop

- ① Schreiben Sie ein Playbook, in dem sichergestellt wird, dass mehrere Dienste vorhanden und gestartet sind.
- ② Schreiben Sie ein Playbook, in dem bestimmte Benutzer und Dateien angelegt werden.
- ③ Schreiben Sie ein Playbook, in dem sichergestellt wird, dass bestimmte Programme installiert und deren Dienste gestartet sind.

1. Schreiben Sie ein Playbook, in dem mit dem Modul `ansible.builtin.service` und einem Loop sichergestellt wird, dass auf allen Systemen der Gruppe `suse` die Dienste `sshd`, `cron` und `systemd-timedated` gestartet sind.
2. Schreiben Sie ein Playbook, in dem in einem ersten Task mit dem Modul `ansible.builtin.user` drei Benutzer angelegt werden. Lassen Sie die Benutzernamen mit `loop` aus einer in einer Variable deklarierten Liste auslesen. Erstellen Sie einen weiteren Task, in dem mit dem Modul `ansible.builtin.file` ein jeweils nach den Benutzern benanntes Verzeichnis unter dem Pfad `/root/` angelegt wird. Lassen Sie auch hierfür mit `loop` die Liste mit den Benutzernamen auslesen.
3. Schreiben Sie ein Playbook, in dem in einem ersten Task sichergestellt wird, dass die Programme `ntp` und `apache2` in der aktuellsten Version auf dem Zielsystem `ubuntu1` vorhanden sind. Stellen Sie in einem weiteren Task mit dem `ansible.builtin.service`-Modul sicher, dass die zu den Programmen zugehörigen Dienste gestartet wurden.

## Musterlösung

- Schreiben Sie ein Playbook, in dem mit dem Modul `ansible.builtin.service` und einem Loop sichergestellt wird, dass auf allen Systemen der Gruppe `suse` die Dienste `sshd`, `cron` und `systemd-timedated` gestartet sind:

Erstellen Sie am Ende des Tasks eine Liste, durch die `loop` iterieren kann:

```
---
- name: a playbook, that should make sure with a loop that
  ↪ several services are running
  hosts: suse
  tasks:
    - name:
      ansible.builtin.service:
        name: "{{ item }}"
        state: started
    loop:
      - 'sshd'
      - 'cron'
      - 'systemd-timedated'
```

Das aufgerufene Playbook:

```
# ansible-playbook -i /root/code/inventory stateloop.yml

PLAY [a playbook, that should make sure with a loop that several
  ↪ services are running] **

TASK [Gathering Facts] **
ok: [suse1]
ok: [suse2]

TASK [service] **
ok: [suse1] => (item=sshd)
ok: [suse2] => (item=sshd)
ok: [suse1] => (item=cron)
ok: [suse2] => (item=cron)
changed: [suse1] => (item=systemd-timedated)
changed: [suse2] => (item=systemd-timedated)

PLAY RECAP **
suse1          : ok=2    changed=1    unreachable=0    failed=0
suse2          : ok=2    changed=1    unreachable=0    failed=0
```

- Schreiben Sie ein Playbook, in dem in einem ersten Task mit dem Modul `ansible.builtin.user` drei Benutzer angelegt werden. Lassen Sie die Benutzernamen mit `loop` aus einer in einer Variable deklarierten Liste auslesen. Erstellen Sie einen weiteren Task, in dem mit dem Modul `ansible.builtin.file` ein jeweils nach den Benutzern benanntes Verzeichnis unter dem Pfad `/root/` angelegt wird. Lassen Sie auch hierfür mit `loop` die Liste mit den Benutzernamen auslesen:

Erstellen Sie eine Liste als Variable, durch die `loop` für beide Tasks iterieren kann:

```
---
- name: loop with var list
  hosts: centos2
  vars:
    somelist:
      - user1
      - user2
      - user3
  tasks:
    - name: make sure users are there
      ansible.builtin.user:
        name: "{{ item }}"
        state: present
      loop: "{{ somelist }}"
    - name: make sure a folder named after each user from the
      ↪ list is created
      ansible.builtin.file:
        path: "/root/{{ item }}"
        state: directory
      loop: "{{ somelist }}"
```

Beiden Tasks ist jeweils ein `loop` angefügt, das dafür sorgt, dass die Tasks sich wiederholen und dabei die in den Variablen deklarierte Liste iterieren. Für jeden Punkt in der Liste werden die Tasks erneut ausgeführt. Somit werden im ersten Task drei Benutzer und im zweiten Task drei Verzeichnisse angelegt.

```
# ansible-playbook -i /root/code/inventory loop_var_list1.yml

PLAY [loop with var list] **

TASK [Gathering Facts] **
ok: [centos2]

TASK [make sure users are there] **
ok: [centos2] => (item=user1)
ok: [centos2] => (item=user2)
ok: [centos2] => (item=user3)

TASK [make sure a folder named after each user from the list is
      ↪ created] **
ok: [centos2] => (item=user1)
ok: [centos2] => (item=user2)
ok: [centos2] => (item=user3)

PLAY RECAP **
centos2 : ok=3     changed=0     unreachable=0     failed=0
```

3. Schreiben Sie ein Playbook, in dem in einem ersten Task sichergestellt wird, dass die Programme `ntp` und `apache2` in der aktuellsten Version auf dem Ziel-  
system `ubuntul` vorhanden sind. Stellen Sie in einem weiteren Task mit dem  
`ansible.builtin.service`-Modul sicher, dass die zu den Programmen zuge-  
hörigen Dienste gestartet wurden:

Eine mögliche Lösung könnte wie folgt aussehen:

```
---
- name: A playbook that to ensure apache2 and ntp are present and
  ↪ running
  hosts: ubuntul
  vars:
    packagelist:
      - apache2
      - ntp
  tasks:
    - name: ensure that packages are there in their latest version
      ansible.builtin.package:
        name: "{{ packagelist }}"
        state: latest

    - name: ensure that the services are up and running
      ansible.builtin.service:
        name: "{{ item }}"
        state: started
      loop: "{{ packagelist }}"
```

In diesem Playbook wird zunächst eine Liste der Pakete in der Variable `packagelist` deklariert. Diese Variable wird im ersten Task direkt an das `ansible.builtin.package`-Modul übergeben. Im zweiten Task wird mit dem `ansible.builtin.service`-Modul und einem `loop`, der über die Liste aus der Variable `packagelist` iteriert, sichergestellt, dass die Dienste gestartet sind:

```
# ansible-playbook -i inventory loop_exercise3.yml

PLAY [A playbook that to ensure apache2 and ntp are present and
  ↪ running] **

TASK [Gathering Facts] **
ok: [ubuntul]

TASK [ensure that packages are there in their latest version] **
changed: [ubuntul]

TASK [ensure that the services are up and running] **
ok: [ubuntul] => (item=apache2)
ok: [ubuntul] => (item=ntp)

PLAY RECAP **
ubuntul : ok=3    changed=1    unreachable=0
          ↪ failed=0     skipped=0    rescued=0    ignored=0
```

## 16 Ansible Playbooks und Roles



# Ansible Playbooks und Roles

## 16.1 Zielsetzung



# Zielsetzung

Dieses Kapitel liefert einen tieferen Einblick in Playbooks:

- Roles
- Jinja2-Syntax
- Nutzung von fortgeschrittenen Variablen
- fortgeschrittene Konditionale
- fortgeschrittene Schleifen
- Ansible Galaxy

## 16.2 Roles



# Was sind Rollen?

- Strukturierung von Playbooks nach Schema
- Auslagerung von:
  - Tasks
  - Vars
  - Handlers
  - Files
  - Templates

Rollen (*Roles*) haben in Ansible die Funktion, einmal geschriebene Anweisungen, Tasks und Plays in einem größeren Maßstab verwendbar und wieder verwendbar zu machen. Das Prinzip ist es, nicht mehr einen Task für eine bestimmte Aufgabe, z. B. das Installieren eines Webservers, in einem Playbook zu beschreiben, sondern diese Aufgabe in einer Rolle zu definieren. Dies hat zum einen den Vorteil, dass diese Rolle auf verschiedene Systeme und in verschiedenen Kontexten angewandt werden kann, und zum anderen, dass auch nur Teile verschiedener Rollen genutzt und kombiniert werden können.

Das Anwendungsziel ist es nicht mehr, einzelne Playbooks für einzelne Zielsysteme zu schreiben, sondern Zielsystemen oder auch ganzen Systemlandschaften eine bestimmte Rolle zuzuweisen, wie z. B. die eines Webservers. Ansible sorgt dann anhand der vordefinierten Rollen dafür, dass die Systeme dem beschriebenen Zustand entsprechen. Zugleich steigt mit der Verwendung von Rollen die Interoperabilität, da Rollen in verschiedenen Kontexten und von verschiedenen Abteilungen genutzt und untereinander ausgetauscht werden können.

Rollen werden dafür in Playbooks eingebunden und als Teil von diesen ausgeführt. Somit können verschiedene Rollen in ein und dasselbe Playbook eingebunden und miteinander aufgerufen werden.

## Warum Rollen?

- Übersichtlichkeit
- Wiederverwendbarkeit von Code
- Trennung von Code und Daten
- Flexibilität für Systeme
- Default-Format, um Ansible-Code zu teilen

Die Erweiterung des eigenen Codes von Playbooks auf Rollen wird spätestens dann attraktiv, wenn die Aufgaben im Playbook sehr umfangreich und das Playbook somit sehr lang und unübersichtlich wird. In Rollen lagern Sie Ihre Tasks aus dem Playbook in eine übersichtlichere Task-Struktur der Rolle aus, sodass Tasks anhand ihrer Funktion gruppiert werden können. Zudem bieten Rollen durch die Art ihrer Verfassung eine große Wiederverwendbarkeit, denn ihr Aufbau ist so angelegt, dass sie ein Gerüst an Funktionalität bieten, das von den Anwendern durch Variablen angepasst werden kann. Die eigentliche Hauptaufgabe der Rolle bleibt dabei immer gleich und muss im Nachhinein im Prinzip nicht mehr angerührt werden. Des Weiteren bieten Rollen in ihrer Struktur die Möglichkeit, bestimmte Dateien direkt mit auszuliefern, was das Anlegen von z. B. Konfigurationsdateien automatisiert ermöglicht und somit eine gute Trennung von Code und Daten bedeutet. Die Rolle bildet den Code ab und die Daten werden über Variablen in diesen eingegeben. Somit bieten Rollen ein hohes Maß an Flexibilität gegenüber den Systemen, auf denen sie zum Einsatz kommen, und sind nicht zuletzt durch ihre kompakte Verzeichnisstruktur höchst portabel. So lassen sich Rollen auch sehr gut in Versionierungssystemen wie Git verwalten und teilen.

## 16.3 Struktur von Rollen



# Struktur von Rollen

**Verzeichnisstruktur**

```

roles
|-- httpd
    |-- defaults
    |-- files
    |-- handlers
    |-- meta
    |-- tasks
    |-- templates
    |-- tests
    |-- vars

```

---

B1 Systems GmbH      Terraform & Ansible Grundlagen      Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de      195 / 279

Die Struktur der Rollen wird in einer Verzeichnisstruktur abgebildet, innerhalb derer die verschiedenen Komponenten an- und abgelegt werden. Dabei wird zunächst die Rolle selbst festgelegt und bekommt ein Verzeichnis, in dem im Folgenden die Komponenten verwaltet werden. Als Komponenten werden die einzelnen Teile eines Playbooks in separaten Dateien und Verzeichnissen abgelegt, damit sich diese in der Rolle einbinden lassen und modular verfügbar sind. Dabei ist zu beachten, dass Rollen immer auf dem Controller liegen und ausgewertet werden, auf dem Ansible ausgeführt wird, und die Änderungen im Anschluss auf die Zielsysteme übertragen werden.

### Die Verzeichnisstruktur einer Rolle:

```

roles
|-- httpd
    |-- defaults
    |-- files
    |-- handlers
    |-- meta
    |-- tasks
    |-- templates
    |-- tests
    |-- vars

```

Ansible sucht standardmäßig im Verzeichnis `roles` nach Rollen, wenn diese in einem Playbook aufgerufen werden. In diesem Verzeichnis liegen Unterverzeichnisse, die die eigentlichen Rollen beinhalten. Im Beispiel gibt es ein Verzeichnis „`httpd`“. Hier finden sich dann weitere Unterverzeichnisse, in denen die einzelnen Komponenten der Rolle angelegt sind:

### **tasks**

In diesem Verzeichnis befinden sich die YAML-Dateien, die die Tasks der Rolle beschreiben. Dabei ist lediglich die `main.yml` obligatorisch, alle weiteren Dateien sind optional.

### **vars**

Hier werden die in der Rolle angewandten Variablen in YAML-Dateien deklariert. Dabei ist auch hier lediglich die `main.yml` obligatorisch, alle weiteren Dateien sind optional.

### **defaults**

Im Verzeichnis `defaults` können Sie unspezifische Variablen in der `main.yml` deklarieren. Diese Variablen kommen immer dann in der Rolle zur Anwendung, wenn diese nicht spezifisch an anderer Stelle deklariert sind.

### **files**

Im `files`-Verzeichnis liegen Dateien, die von der Rolle verwendet werden. Das sind z. B. Konfigurationsdateien, die auf die Zielsysteme kopiert werden sollen.

### **templates**

Das `templates`-Verzeichnis dient dem Ablegen von Dateien, die vom Modul `template` mit der Templating-Sprache `Jinja2` ausgewertet werden sollen. Dazu gehören z. B. Dateien mit Informationen wie Hostnamen oder IP-Adressen, die dynamisch aus Variablen oder `ansible_facts` ausgelesen werden sollen.

### **meta**

In `meta` werden spezifische Dateien für Ansible-Galaxy wie Lizenzdateien usw. abgelegt.

### **handlers**

Auch das Verzeichnis `handlers` beinhaltet eine `main.yml`, die etwaige Handlers der Rolle beschreibt.

Die Verzeichnisse `defaults`, `tasks`, `vars`, `meta` und `handlers` enthalten jeweils eine `main.yml`-Datei, die standardmäßig von Ansible ausgewertet wird. Die hier gezeigte Verzeichnisstruktur dient der Anschauung einer Möglichkeit. In der Praxis ist es nicht immer nötig, jedes der hier gezeigten Verzeichnisse anzulegen oder zu nutzen. Die folgenden Beispiele zeigen, dass sich die Dateistruktur in den Unterverzeichnissen weiter verfeinern kann und damit die Möglichkeit gegeben ist, die einzelnen Komponenten der Rolle sehr detailliert zu gestalten und festzulegen.

### 16.3.1 Aufgabenteil



## Aufgabenteil – Verzeichnisstruktur von Rollen

Schreiben Sie ein Playbook, das in /root/code/roles die vollständige Verzeichnisstruktur für eine Rolle apache2\_role anlegt. Lassen Sie das Playbook – dort wo nötig – auch die main.yml-Dateien anlegen.

Schreiben Sie ein Playbook, das in /root/code/roles die vollständige Verzeichnisstruktur für eine Rolle apache2\_role anlegt. Lassen Sie das Playbook – dort wo nötig – auch die main.yml-Dateien anlegen.

## Musterlösung

Schreiben Sie ein Playbook, das in `/root/code/roles` die vollständige Verzeichnisstruktur für eine Rolle `apache2` anlegt. Lassen Sie das Playbook – dort wo nötig – auch die `main.yml`-Dateien anlegen:

Es gibt mehrere Möglichkeiten, ein solches Playbook zu verfassen. Eine mögliche Lösung könnte wie folgt aussehen:

```
---
- name: to let a role structure on this computer exist
  hosts: localhost
  vars:
    roledir: "/root/code/roles/apache2_role"
    folderlist:
      - 'tasks'
      - 'vars'
      - 'defaults'
      - 'files'
      - 'templates'
      - 'meta'
      - 'handlers'
  tasks:
    - name: first task to make sure the directories exist
      ansible.builtin.file:
        path: "{{ roledir }}/{{ item }}"
        state: directory
      loop: "{{ folderlist }}"
    - name: second task to ensure the main.yml is present where useful
      ansible.builtin.file:
        path: "{{ roledir }}/{{ item }}/main.yml"
        state: touch
      loop: "{{ folderlist }}"
      when: item in [ 'tasks', 'vars', 'defaults', 'meta', 'handlers' ]
    - name: now to have some minus in the main.yml
      ansible.builtin.lineinfile:
        path: "{{ roledir }}/{{ item }}/main.yml"
        line: "----"
        state: present
      loop: "{{ folderlist }}"
      when: item == 'tasks' or item == 'vars' or item == 'defaults'
         or item == 'meta' or item == 'handlers'
```

Als Host muss `localhost` angegeben werden, da die Rolle auf dem Controller liegt und nicht auf einem der Zielsysteme. In diesem Playbook ist der Pfad der Rolle in der Variablen `roledir` deklariert, damit er wiederverwendbar wird. Die Namen der einzelnen Rollenverzeichnisse sind in einer Variablen `folderlist` als Liste deklariert, damit die Tasks diese mit der `loop`-Schleife iterieren können.

Der erste Task legt die Verzeichnisse an. Dafür iteriert `loop` die Liste in der Variablen `folderlist` mit dem Modul `file` und der Option `state:directory`, wodurch al-

le Verzeichnisse angelegt werden. Das Modul `file` legt dabei alle im Pfad angegebenen Verzeichnisse an, sofern diese noch nicht vorhanden sind. Auch das Verzeichnis `/root/code/roles` muss also nicht zusätzlich angelegt werden.

Der zweite Task erstellt nun mit dem Modul `file` und der Option `state:touch` eine Datei `main.yml` in den Verzeichnissen der Liste in der Variablen `folderlist`, sofern die `when`-Kondition zutrifft. Diese prüft, ob das von `loop` iterierte Listenobjekt in der Liste steht, die unter `when` in der kompakten YAML-Schreibweise angegeben ist. Eine andere Möglichkeit wäre es, diese Liste ebenfalls als Variable zu deklarieren und den Vergleich mit „`when: item in $Variable`“ durchzuführen.

Der dritte und letzte Task schreibt mit dem Modul `lineinfile` die drei Minuszeichen in die `main.yml`-Dateien, damit diese zu gültigen YAML-Dateien werden. Hier hätte auch dieselbe `when`-Kondition wie im zweiten Task verwendet werden können. Aus Anschauungsgründen wird hier jedoch eine leicht abgeänderte Bedingung gesetzt, in der die zu vergleichenden Listenpunkte separat mit dem Operator `or` abgefragt werden.

Wird das Playbook ausgeführt, legt es die Verzeichnisse an und erstellt in allen Verzeichnissen außer `files` und `templates` eine `main.yml`, in die es --- einträgt:

```
# ansible-playbook -i /root/code/inventory role.yml

PLAY [to let a role structure on this computer exist] **

TASK [Gathering Facts] **
ok: [localhost]

TASK [first task to make sure the directories exist] **
changed: [localhost] => (item=tasks)
changed: [localhost] => (item=vars)
changed: [localhost] => (item=defaults)
changed: [localhost] => (item=files)
changed: [localhost] => (item=templates)
changed: [localhost] => (item=meta)
changed: [localhost] => (item=handlers)

TASK [second task to ensure the main.yml is present where useful] **
changed: [localhost] => (item=tasks)
changed: [localhost] => (item=vars)
changed: [localhost] => (item=defaults)
skipping: [localhost] => (item=files)
skipping: [localhost] => (item=templates)
changed: [localhost] => (item=meta)
changed: [localhost] => (item=handlers)

TASK [now to have some minus in the main.yml] **
changed: [localhost] => (item=tasks)
changed: [localhost] => (item=vars)
changed: [localhost] => (item=defaults)
skipping: [localhost] => (item=files)
skipping: [localhost] => (item=templates)
changed: [localhost] => (item=meta)
changed: [localhost] => (item=handlers)
```

## 16 Ansible Playbooks und Roles

```
PLAY RECAP **  
localhost : ok=4      changed=3      unreachable=0  
          ↳ failed=0
```

Das Rollenverzeichnis sieht nun wie folgt aus:

```
# tree /root/code/roles/apache2_role  
/root/code/roles/apache2_role  
|-- defaults  
|   '-- main.yml  
|-- files  
|-- handlers  
|   '-- main.yml  
|-- meta  
|   '-- main.yml  
|-- tasks  
|   '-- main.yml  
|-- templates  
`-- vars  
    '-- main.yml
```

7 directories, 5 files

### 16.3.2 Rollen: tasks



## Rollen: tasks

```
roles/httpd/tasks/
  tasks/
    main.yml
    config.yml
    modules.yml
    vhost.yml
```

B1 Systems GmbH      Terraform & Ansible Grundlagen      197 / 279

In Verzeichnis `tasks` werden die Dateien angelegt, die für Tasks von Bedeutung sind. Obligatorisch ist dabei die Datei `main.yml`, in der die für die Rolle abzuarbeitenden Tasks enthalten sind. Neben der `main.yml` kann das Verzeichnis `tasks` aber auch weitere YAML-Dateien enthalten, in denen verschiedene Tasks für bestimmte Aufgaben stehen. Diese weiteren Dateien können dann mit einer `- include_tasks:-`-Anweisung in der `main.yml` eingebunden werden. Auf diese Art kann die Rolle flexibel um einzelne Tasks erweitert und wieder verkürzt werden.

## Rollen: tasks

### tasks/main.yml

```
---
- name: Install httpd
  ansible.builtin.package:
    name: "{{ httpd_package }}"
    state: "{{ httpd_package_state }}"

- include_tasks: modules.yml
- include_tasks: config.yml
- include_tasks: vhost.yml
```

Das Format der Einträge für Tasks in der main.yml-Datei im tasks-Verzeichnis entspricht dem Format von Tasks im Playbook:

```
---
- name: ensure the latest version of nano is present
  ansible.builtin.package:
    name: nano
    state: latest

- name: test task
  debug:
    msg: "This is a test."
```

Ebenso ist es möglich, Variablen zu nutzen oder Dateien einzubinden, die weitere Tasks enthalten. Ein Beispiel einer solchen komplexeren main.yml könnte so aussehen:

```
---
- name: Install httpd
  ansible.builtin.package:
    name: "{{ httpd_package }}"
    state: "{{ httpd_package_state }}"

- include_tasks: modules.yml
- include_tasks: config.yml
- include_tasks: vhost.yml
```

Hier werden vom Task Variablen aufgerufen, die entweder in `roles/httpd/defaults/` oder in `roles/httpd/vars/` deklariert sind. Zudem werden über die Anweisung `“- include_tasks:”` Dateien mit weiteren Tasks eingelesen und die darin enthaltenen Tasks mit ausgeführt.



### 16.3.3 Aufgabenteil



## Aufgabenteil – Tasks in Rollen

Überführen Sie die folgenden Tasks in die Task-Struktur einer Rolle für apache2.

Überführen Sie die folgenden Tasks in die Task-Struktur einer Rolle für apache2:

```
- name: ensure that packages are there in their latest version
  ansible.builtin.package:
    name:
      - 'apache2'
      - 'openssh'
    state: latest

- name: ensure that the services are up and running
  ansible.builtin.service:
    name: "{{ item }}"
    state: started
  loop:
    - 'apache2'
    - 'sshd'
```

## Musterlösung

**Überführen Sie die folgenden Tasks in die Task-Struktur einer Rolle für apache2:**

Legen Sie für die Lösung eine Datei

/root/code/roles/apache2\_role/tasks/main.yml an.

Da die Rolle ausschließlich die Funktion hat, apache2 zu installieren, wird hier auf das Paket openssh verzichtet. Anschließend muss daher auch nur der Dienst apache2 gestartet werden:

```
---
- name: ensure that packages are there in their latest version
  ansible.builtin.package:
    name: apache2
    state: latest

- name: ensure that the service is up and running
  ansible.builtin.service:
    name: apache2
    state: started
```

Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de

#### 16.3.4 Rollen: handlers



## Rollen: handlers

```
roles/httpd/handlers
handlers/
    main.yml
```

```
handlers/main.yml
- name: restart httpd
  ansible.builtin.service:
    name: "{{ httpd_service }}"
    state: restarted
```

B1 Systems GmbH      Terraform & Ansible Grundlagen      200 / 279

Bei Änderung oder Aktualisierung eines Pakets oder einer Konfigurationsdatei eines Dienstes mit Ansible wird diese Änderung erst beim Neustart des Dienstes aktiv. Diese Aufgabe übernehmen die *Handlers*, die die Dienste im Falle einer Veränderung neu starten können. Dies erfolgt dann, wenn der Task ein `changed` zurückgibt. So kann die Idempotenz erhalten bleiben, d. h. dass Dienste auch bei mehrfachem Ausführen der Tasks im Falle von Änderungen weiterlaufen.

Der folgende Task wird mehrfach ausgeführt:

```
---
- name: ensure that packages are there in their latest version
  ansible.builtin.package:
    name: openssh
    state: latest
```

Erfolgt eine Änderung am Paket `openssh`, weil z. B. eine neuere Version vorliegt, so muss der Dienst `sshd` neu gestartet werden, damit die Änderungen wirksam werden. In diesem Fall wird dem Task ein `notify`: angehängt, welches dafür sorgt, dass der Dienst die gewünschte Handlung vollzieht:

```
---
- name: ensure that packages are there in their latest version
```

## 16 Ansible Playbooks und Roles

```
ansible.builtin.package:
  name: openssh
  state: latest
  notify: restart sshd
```

Das `notify:` ruft den Handler `restart sshd` auf. Damit dieser Handler aufgerufen werden kann, muss er in der `main.yml` im Verzeichnis `/handlers` eingetragen sein:

```
---
- name: restart sshd
  ansible.builtin.service:
    name: sshd
    state: restarted
```

Nun wird die Rolle jedes Mal bei Ausführung des Task, wenn ein `changed` zurückgegeben wird, auch den Handler aufrufen, der den Dienst neu startet. Der Task sucht entsprechend des Verweises unter `notify:` in der `main.yml` im Verzeichnis `/roles/httpd/handlers/` nach dem richtigen Eintrag und wendet den Handler an, sobald der Task ein `changed` zurück gibt.

### Handlers direkt im Playbook einsetzen

Handlers stehen in Ansible auch außerhalb von Rollenstrukturen zu Verfügung und können auch direkt in Playbooks verwendet werden. Damit Handlers in Playbooks genutzt werden können, müssen diese im Playbook definiert sein:

```
---
- name: a playbook with a handler
  hosts: ubuntu1
  handlers:
    - name: restart sshd
      ansible.builtin.service:
        name: sshd
        state: restarted
  tasks:
    - name: make sure openssh-server is installed in the latest version
      ansible.builtin.package:
        name: openssh-server
        state: latest
      notify: restart sshd
    - name: make sure sshd is started in the first place
      ansible.builtin.service:
        name: sshd
        state: started
```

Hier wird der Handler direkt im Playbook definiert und am Ende des Plays abgerufen, falls der Task mit dem `notify` mit einem `changed` beendet wurde, das Paket `openssh-server` also auf eine neuere Version aktualisiert wurde.

### Service-Task zusätzlich zum Handler

Es empfiehlt sich, an dem Task, der den Dienst verändert, zusätzlich zum Handler noch einen weiteren Service-Task einzurichten, der mit der Option `state: started` den Dienst startet. Da der Handler den Dienst nur startet, wenn der Task ein `changed` zurückgibt, hat er keinen Einfluss auf den Dienst, wenn sich keine Änderung vollzogen hat. Soll allerdings sichergestellt werden, dass der Dienst nach dem Ausführen von `ansible` in jedem Fall gestartet wurde, so kann dies nur das `ansible.builtin.service`-Modul in einem eigenen Task leisten. Ein Szenario für so einen Fall wäre z.B., dass der Dienst durch einen anderen Dienst oder eine Person gestoppt wurde. Wird nun `ansible` ausschließlich mit einem Handler ausgeführt und an dem Dienst tritt keine Änderung ein, so bleibt er gestoppt. Damit der Dienst nach dem Ausführen von `ansible` läuft, muss also ein zusätzlicher Task mit dem `ansible.builtin.service`-Modul gestartet werden. Dieser Task kann allerdings mit einer `when`-Kondition davon abhängig gemacht werden, dass der vorherige Task kein `changed` zurückgegeben hat.

## Mehrere Handlers aktivieren

Notify eine Liste von Handlers übergeben

```
tasks:  
  - name: update config  
    ansible.builtin.copy:  
      src: "{{ item['source'] }}"  
      dest: "{{ item['path'] }}"  
    loop: "{{ webstack_config }}"  
    notify:  
      - restart httpd  
      - restart appserver
```

Manchmal kann es nützlich sein, nach dem Ausführen eines Tasks mehrere Dienste neu zu starten, z. B. wenn in dem Task Konfigurationsdateien geändert werden, die sich auf mehrere Dienste auswirken. In diesem Fall können mit `notify` einfach mehrere Handlers in einer Liste angegebenen werden:

```
tasks:  
  - name: update config  
    ansible.builtin.copy:  
      src: "{{ item['source'] }}"  
      dest: "{{ item['path'] }}"  
    loop: "{{ webstack_config }}"  
    notify:  
      - restart httpd  
      - restart appserver
```

Damit die Dienste `httpd` und `appserver` auch neu gestartet werden können, müssen die Handlers in der `handlers/main.yml` der Rolle oder im Playbook selbst eingetragen sein:

```
handlers:
  - name: restart httpd
    ansible.builtin.service:
      name: 'httpd'
      state: restarted

  - name: restart appserver
    ansible.builtin.service:
      name: 'tomcat'
      state: restarted
```

Wenn nun der Task ausgeführt wird und ein `changed` zurück gibt, werden die beiden Dienste `httpd` und `appserver` neu gestartet.



### Hinweis Reihenfolge

Die Handlers werden immer in der Reihenfolge ausgeführt, in der sie in der `handlers/main.yml` definiert wurden und nicht in der Reihenfolge, in der sie bei `notify:` stehen.

# Handlers in jedem Fall

Handlers in jedem Fall ausführen:

```
--force-handlers auf der Kommandozeile
# ansible-playbook handler_playbook.yml --force-handlers
```

## force\_handlers im Playbook

```
---
- name: a playbook with a handler
  hosts: ubuntu1
  handlers:
    - name: restart sshd
      ansible.builtin.service:
        name: sshd
        state: restarted
  force_handlers: true
```

Schlägt ein Task fehl, an dessen Ende mit `notify` ein Handler ausgeführt werden sollte, würde standardmäßig der Handler nicht gestartet. Sie können aber dafür sorgen, dass der Handler auf jeden Fall ausgeführt wird, d. h. auch bei Scheitern des Task.

Auf der Kommandozeile verwenden Sie die Option `--force-handlers` in Kombination mit `ansible-playbook`, um dafür zu sorgen, dass die im Playbook enthaltenen Handlers in jedem Fall ausgeführt werden:

```
# ansible-playbook handler_playbook.yml --force-handlers
```

In Playbooks können Sie einzelne Plays mit `force_handlers: true` aktivieren, sodass die im Play enthaltenen Handlers auch bei Scheitern der Tasks ausgeführt werden:

```
---
- name: a playbook with a handler
  hosts: ubuntu1
  handlers:
    - name: restart sshd
      service:
        name: sshd
        state: restarted
  force_handlers: true
```



# Handlers gezielt ausführen

Handlers an bestimmten Punkten ausführen:

- nicht erst am Ende des Playbooks
- z. B. direkt nach Task mit `notify`
- per meta-Action: `flush_handlers`
- für Tasks die ein *changed* zurückgeben

Handler per meta-Action ausführen:

```
- name: make sure service is started
  ansible.builtin.service:
    name: apache2
    state: started
    notify: apache2 reload
- name: force all notified handlers to run at this point
  meta: flush_handlers
```

Da es wünschenswert sein kann, die über `notify` eingesetzten Handlers nicht erst am Ende des Playbooks, sondern schon während eines Durchlaufs zu aktivieren, besteht die Möglichkeit dies zu erzwingen. Mit dem Modul `ansible.builtin.meta` können Sie die Aktion `flush_handlers` ausführen, die bewirkt, dass alle bis zu diesem Zeitpunkt per `notify` aktivierte Handlers mit dem Aufruf der Aktion ausgeführt werden, anstatt erst zum Ende des Playbooks. Dafür erzeugen Sie einen Task, der mit dem `ansible.builtin.meta`-Modul die Aktion `flush_handlers` ausführt:

```
- name: make sure service is started
  ansible.builtin.service:
    name: apache2
    state: started
    notify: apache2 reload
- name: force all notified handlers to run at this point
  meta: flush_handlers
```

Bereits ausgeführte Handlers werden im Falle eines weiteren meta-Tasks nicht erneut ausgeführt, sodass es bspw. möglich wäre, jeden gesetzten Handler mit einem eigenen `flush_handlers` direkt auszuführen.

# Handlers unter Oberbegriffen zusammenfassen

## Handlers unter Oberbegriffen zusammenfassen

```
- handlers:
  - name: restart httpd
    ansible.builtin.service:
      name: 'httpd'
      state: restarted
      listen:
        - 'webstack'
        - 'webserver'
  - name: restart dns
    ansible.builtin.service:
      name: 'pdns'
      state: restarted
      listen: 'webstack'
```

Die Handlers in der Datei `handlers/main.yml` können neben ihrem Namen auch per `listen:`-Direktive unter Oberbegriffen zusammengefasst und über diese aufgerufen werden. Es ist möglich, mehrere Handlers mit demselben Oberbegriff zu versehen und so über diesen Oberbegriff gemeinsam zu starten. Die Oberbegriffe werden den `handlers` mit der Option `listen:` zugewiesen:

```
- name: restart httpd
  ansible.builtin.service:
    name: 'httpd'
    state: restarted
  listen:
    - 'webstack'
    - 'webserver'

- name: restart dns
  ansible.builtin.service:
    name: 'pdns'
    state: restarted
  listen: 'webstack'
```

Es können einem Handler beliebig viele dieser Oberbegriffe zugeordnet werden. Im Beispiel sind dem Handler `restart httpd` die beiden Oberbegriffe `webstack` und `webserver` zugeordnet, und dem Handler `restart dns` neben seinem Namen noch

der Oberbegriff `webstack`. Nun können die Handlers in einem Task auch mit diesen Oberbegriffen aufgerufen werden:

```
- name: a task to update httpd
  ansible.builtin.package:
    name: "{{ httpd_package }}"
    state: present
    notify: 'webstack'
```

In diesem Fall werden die beiden Dienste `httpd` und `pdns` mit dem `notify: 'webstack'` neu gestartet, sobald der Task ein `changed` zurück gibt.



### 16.3.5 Aufgabenteil



## Aufgabenteil – Handlers

Erstellen Sie einen Handler für den Task in der Rolle apache2\_role, der den Dienst neu startet, sofern der Task das Paket aktualisiert hat.

Erstellen Sie einen Handler für den Task in der Rolle apache2\_role, der den Dienst neu startet, sofern der Task das Paket aktualisiert hat.

## Musterlösung

**Erstellen Sie einen Handler für den Task in der Rolle `apache2_role`, der den Dienst neu startet, sofern der Task das Paket aktualisiert hat:**

Tragen Sie den Handler in die Datei

/root/code/roles/apache2\_role/handlers/main.yml ein:

```
---
- name: restart webserver
  ansible.builtin.service:
    name: apache2
    state: restarted
```

Ergänzen Sie im Anschluss das `notify: restart webserver` in Ihrem Task in der /root/code/roles/apache2\_role/tasks/main.yml:

```
---
- name: ensure that packages are there in their latest version
  ansible.builtin.package:
    name: apache2
    state: latest
  notify: restart webserver

- name: ensure that the service is up and running in the first place
  ansible.builtin.service:
    name: apache2
    state: started
```

### 16.3.6 Rollen: `vars`, `defaults`



## Rollen: `vars`, `defaults`

Zwei Orte um Variablen einer Rolle zu bestimmen:

- `vars`: Variablen für Funktion der Rolle
- `defaults`: Variablen für Konfiguration der Rolle

### `roles/httpd/vars`

```
vars/
  main.yml
  RedHat.yml
  Suse.yml
```

### `roles/httpd/defaults`

```
defaults/
  main.yml
```

Ansible hält für Rollen mehrere Orte und Möglichkeiten bereit, Variablen zu deklarieren. Diese Möglichkeiten unterscheiden sich allerdings in ihrer Qualität und in ihrem Zweck. Typischerweise werden Variablen für Rollen in der Datei `defaults/main.yml` und in YAML-Dateien im Verzeichnis `vars/` deklariert.

#### Variablen in der Datei `defaults/main.yml`

In der Datei `defaults/main.yml` deklarierte Variablen sind für die gesamte Rolle gültig. Die hier in den Variablen deklarierten Werte dienen als leicht zu überschreibende Standardvorgaben. Diese Standardvorgaben können von den Anwendern der Rolle leicht überschrieben werden um angepasste Werte einzusetzen und die Rolle somit zu konfigurieren. Ein Beispiel für eine solche Variable in der `defaults/main.yml` wäre der folgende Eintrag:

```
---
httpd_package_state: present
```

Ist diese Variable in der `defaults/main.yml` deklariert, so kann sie in der Rolle verwendet werden.

## 16 Ansible Playbooks und Roles

In der Datei tasks/main.yml kann nun die Zeile state: present

```
---
```

- name: Ensure httpd is installed in its present state  
  ansible.builtin.package:  
    name: httpd  
    state: present

zu state: httpd\_package\_state geändert werden, da Ansible die Variable aus der defaults/main.yml auslesen kann:

```
---
```

- name: Ensure httpd is installed in its {{ httpd\_package\_state }}  
  ↪ state  
  ansible.builtin.package:  
    name: httpd  
    state: "{{ httpd\_package\_state }}"

Wenn die Rolle in ein Playbook eingebunden und ausgeführt wird, liest ansible-playbook nun den in der Variable deklarierten Wert present aus:

```
# ansible-playbook -i inventory role_httpd_playbook.yml
```

PLAY [a playbook to include the role httpd] \*\*

TASK [Gathering Facts] \*\*  
ok: [centos2]

TASK [httpd : Ensure httpd is installed in its present state] \*\*  
ok: [centos2]

PLAY RECAP \*\*  
centos2 : ok=2     changed=0     unreachable=0  
  ↪ failed=0

Die in der defaults/main.yml deklarieren Standartwerte gelten zunächst für die gesamte Rolle, haben aber die Eigenschaft, leicht durch den Benutzer überschreibbar zu sein. Dies ist z. B. dann sinnvoll, wenn die Rolle für eine bestimmte Gruppe angewendet werden soll, deren Konfiguration von den Standardwerten abweicht. Angenommen der Host centos2 soll immer die aktuellste Paketversion erhalten, so wird die Variable für diesen Host in host\_vars/centos2 deklariert:

```
---
```

- httpd\_package\_state: latest

Anschließend liest ansible-playbook für die Rolle die Variable aus der Datei in host\_vars/ statt aus der defaults/main.yml aus:

```
# ansible-playbook -i inventory role_httpd_playbook.yml
```

PLAY [a playbook to include the role httpd] \*\*

TASK [Gathering Facts] \*\*  
ok: [centos2]

```
TASK [httpd : Ensure httpd is installed in its latest state] **  
ok: [centos2]  
  
PLAY RECAP **  
centos2 : ok=2     changed=0     unreachable=0  
          failed=0
```

Den gleichen Effekt haben Variablen, die in `group_vars` für Gruppen deklariert werden.



### Hinweis *Variablen in der defaults/main.yml*

Die Variablen, die in der `defaults/main.yml` deklariert werden, gelten als Standardwerte für die gesamte Rolle. Allerdings haben Sie die Eigenschaft, sich leicht vom Nutzer ändern, d. h. überschreiben zu lassen. Die in der `defaults/main.yml` deklarierten Variablen sind also die „schwäächsten“ Variablen und werden von Variablen überschrieben, die in den Verzeichnissen `host_-` und `group_vars`, dem `vars`-Verzeichnis der Rolle, im Playbook direkt oder als Extra-Vars deklariert werden.



### 16.3.7 Aufgabenteil



## Aufgabenteil – Variablen in defaults/main.yml

- ➊ Deklarieren Sie die Variable apache2\_package\_state mit dem Wert present in der default/main.yml der Rolle apache2\_role.
- ➋ Deklarieren Sie die Variable apache2\_package\_state mit dem Wert latest in der group\_vars-Datei der Gruppe webserver, /root/code/group\_vars/webserver.yml.

1. Deklarieren Sie die Variable apache2\_package\_state mit dem Wert present in der default/main.yml der Rolle apache2\_role.
2. Deklarieren Sie die Variable apache2\_package\_state mit dem Wert latest in der group\_vars-Datei der Gruppe webserver, /root/code/group\_vars/webserver.yml.

## Musterlösung

1. Deklarieren Sie die Variable `apache2_package_state` mit dem Wert `present` in der `default/main.yml` der Rolle `apache2_role`:

Fügen Sie in der `default/main.yml` die gewünschte Zeile ein:

```
---  
apache2_package_state: present
```

2. Deklarieren Sie die Variable `apache2_package_state` mit dem Wert `latest` in der `group_vars`-Datei der Gruppe `webserver`, `/root/code/group_vars/webserver.yml`:

Fügen Sie in `root/code/group_vars/webserver.yml` die gewünschte Zeile ein:

```
---  
apache2_package_state: latest
```

### 16.3.8 Rollen: vars



## Rollen: vars, defaults

- spezifische Funktionsvariablen im vars/-Verzeichnis
  - können in spezifischen Dateien abgelegt werden
  - allgemeine Variablen in defaults/main.yml

vars/Debian.yml

```
httpd_package: 'apache2'
```

vars/Suse.yml

```
httpd_package: 'apache2'
```

vars/RedHat.yml

```
httpd_package: 'httpd'
```

defaults/main.yml

```
httpd_package_state: present
```

Neben den Variablen, die in der defaults/main.yml sowie den host\_- und group\_vars deklariert werden, enthält die Verzeichnisstruktur von Rollen noch das vars/-Verzeichnis. In diesem Verzeichnis werden alle Variablen deklariert, die für die Rolle von Bedeutung sind, jedoch nicht vom Anwender überschrieben oder verändert werden sollen.

### Variablen in der vars/main.yml

Im vars-Verzeichnis liegt eine main.yml, die standardmäßig ausgelesen wird. In dieser Datei werden für die gesamte Rolle geltende Variablen deklariert, wie z. B. der Pfad zur Konfigurationsdatei der SSH-Dienste der Zielsysteme. Um eine solche global geltende Variable zu deklarieren, muss diese in die vars/main.yml eingetragen werden:

```
---
ssh_config_dir: /etc/ssh
ssh_server_config: "{{ ssh_config_dir }}/sshd_config"
ssh_client_global_config: "{{ ssh_config_dir }}/ssh_config"
```

Die so deklarierten Variablen können fortan in der gesamten Rolle genutzt werden.

### Spezifische Variablen dateien in `vars/`

Wie auch das `tasks`-Verzeichnis kann auch das `vars`-Verzeichnis mehrere Dateien enthalten. Typischerweise handelt es sich bei diesen Dateien neben der `main.yml` um Dateien, in denen spezifische Variablen für bestimmte Anwendungsfälle deklariert sind.

Um solche spezifischen Variablen handelt es sich beispielsweise bei distributionsabhängigen Paketen. Zur Vermeidung von Konflikten bei der Ausführung der Rolle können für jede Distribution eigene Variablen deklariert werden. Dazu werden im `vars/-Verzeichnis` weitere YAML-Dateien für die verschiedenen Distributionen angelegt:

Für Red Hat-Systeme: `vars/RedHat.yml` mit dem Inhalt:

```
---
```

```
httpd_package: httpd
```

Für SUSE-Systeme: `vars/Suse.yml` mit dem Inhalt:

```
---
```

```
httpd_package: apache2
```

Für Debian-Systeme: `vars/Debian.yml` mit dem Inhalt:

```
---
```

```
httpd_package: apache2
```

Die Dateinamen entsprechen dem Ansible Fact `ansible_os_family`, die Variable der jeweiligen Paketbezeichnung des Apache2-Webservers. Sind die Variablen auf diese Weise deklariert, können sie in den Tasks der Rolle verwendet werden.



#### Hinweis Eindeutige Variablennamen in Rollen

Setzen Sie Variablen in Rollen mit eindeutigen, der Rolle entsprechenden Präfixen wie z. B. `httpd_` im Fall eines Webservers. Arbeiten Sie mit mehreren Rollen hat dies zum einen den Vorteil, dass Sie einen Überblick über die einzelnen Variablen behalten, und zum anderen überschreiben Sie nicht aus Versehen Ihre Variablen mit „stärkeren“ Variablen wie z. B. `--extra-vars`.



## Einfügen von Variablen-Dateien

- Modul `include_vars` zum Einbinden von Variablen-Dateien in die `tasks/main.yml`

`tasks/main.yml mit dem Modul include_vars`

```
---
- name: Load a file with variables
  include_vars:
    file: RedHat.yml
```

`tasks/main.yml mit dem Modul in verkürzter Schreibweise`

```
---
- include_vars: RedHat.yml
```

Das Modul `include_vars` kann in einem Task Variablen aus YAML-Dateien einbinden und aufrufbar machen. Dabei kann `include_vars` ganz üblich als Modul in einem Task verwendet werden und mit der Option `file` die Datei mit den Variablen aufrufen:

```
---
- name: Load a file with variables
  include_vars:
    file: RedHat.yml
```

In der verkürzten Schreibweise kann der Modulaufruf auf eine Zeile reduziert werden, da ohne Angabe einer Option „`file`“ voreingestellt ist:

```
# cat tasks/main.yml

---
- include_vars: RedHat.yml

- name: Ensure httpd is installed in its {{ httpd_package_state }}
  ↪ state
  ansible.builtin.package:
    name: "{{ httpd_package }}"
    state: "{{ httpd_package_state }}"
```

Jetzt wird die Variable `httpd_package` in der Option `name:` des `package`-Moduls aus der Datei `RedHat.yml` ausgelesen.

Das Modul `include_vars` kann allerdings auch selber Variablen auslesen, weshalb anstatt eines Dateinamens auch eine Variable angegeben werden kann. Es bietet sich hier an, z. B. auf Ansible Facts zurückzugreifen, um automatisch bestimmte Variablen in bestimmten Fällen laden zu können. Im folgenden Beispiel wird der Dateiname der zu ladenden Variabledatei durch den Ansible Fact `ansible_os_family` ersetzt, um das richtige Webserverpaket für die jeweilige Distribution zu installieren:

```
---
- include_vars: "{{ ansible_os_family }}.yml"

- name: Ensure httpd is installed in its {{ httpd_package_state }}
  state: {{ httpd_package_state }}
  ansible.builtin.package:
    name: "{{ httpd_package }}"
    state: "{{ httpd_package_state }}"
```

Dieser Fact entspricht den Dateinamen im `vars`-Verzeichnis. So lädt `include_vars` in Abhängigkeit der Distribution des Zielsystems die richtige Datei mit den entsprechenden Variablen:

```
# ansible all -i /root/code/inventory -m setup -a
  ↪ "filter=ansible_os_family"
ubuntul | SUCCESS => {
  "ansible_facts": {
    "ansible_os_family": "Debian"
  },
  "changed": false
}
ubuntu2 | SUCCESS => {
  "ansible_facts": {Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de
    "ansible_os_family": "Debian"
  },
  "changed": false
}
suse1 | SUCCESS => {
  "ansible_facts": {
    "ansible_os_family": "Suse"
  },
  "changed": false
}
suse2 | SUCCESS => {
  "ansible_facts": {
    "ansible_os_family": "Suse"
  },
  "changed": false
}
centos1 | SUCCESS => {
  "ansible_facts": {
    "ansible_os_family": "RedHat"
  },
  "changed": false
}
centos2 | SUCCESS => {
  "ansible_facts": {
```

```

    "ansible_os_family": "RedHat"
},
"changed": false
}

```

Die Facts entsprechen den Dateinamen im `vars`-Verzeichnis:

```
# ls vars
Debian.yml  RedHat.yml  Suse.yml  main.yml
```

Diese Vorgehensweise hat den Vorteil, dass nun in der Rolle mehrere Systeme mit verschiedenen Anforderungen, wie z.B. Distributionen, eingebunden werden können. Die Gruppe `webserver`:

```
# cat /root/code/inventory

[ubuntu]
ubuntul
ubuntu2

[centos]
centos1
centos2

[suse]
suse1
suse2

[webserver]
centos2
ubuntul
```

besteht aus zwei Systemen mit unterschiedlichen Distributionen. Wenn nun die Rolle für diese Gruppe ausgeführt wird, wählt Ansible automatisch die richtigen Variablen aus, in diesem Fall den Namen des Paketes. Im Playbook wird unter `hosts:` die Gruppe `webserver` angegeben und mit der Option `roles:` wird die entsprechende Rolle eingebunden:

```
# cat role_httpd_playbook.yml

---
- name: a playbook to include the role httpd
  hosts: webserver
  roles:
    - httpd
```

## 16 Ansible Playbooks und Roles

Die Rolle bindet den Task aus der tasks/main.yml ein:

```
# cat tasks/main.yml

---
- include_vars: "{{ ansible_os_family }}.yml"

- name: A debug task to print the Variable
  ansible.builtin.debug:
    msg: "{{ ansible_os_family }}.yml"

- name: Ensure httpd is installed in its {{ httpd_package_state }}
  ↪ state
  ansible.builtin.package:
    name: "{{ httpd_package }}"
    state: "{{ httpd_package_state }}"
```

Ausführen des Playbook mit der Rolle:

```
# ansible-playbook -i /root/code/inventory role_httpd_playbook.yml

PLAY [a playbook to include the role httpd] **

TASK [Gathering Facts] **
ok: [ubuntu1]
ok: [centos2]

TASK [httpd : include_vars] **
ok: [centos2]
ok: [ubuntu1]

TASK [httpd : A debug task to print the Variable] **
ok: [centos2] => {
  "msg": "RedHat.yml"
}
ok: [ubuntu1] => {
  "msg": "Debian.yml"
}

TASK [httpd : Ensure httpd is installed in its latest state] **
ok: [ubuntu1]
ok: [centos2]

PLAY RECAP **

centos2                  : ok=3      changed=0      unreachable=0
                           ↪ failed=0
ubuntu1                  : ok=3      changed=0      unreachable=0
                           ↪ failed=0
```

Der Task bindet nun über die Variable in include\_vars: die Datei in vars/ ein, die dem Ansible Fact ansible\_os\_family der jeweiligen Distribution entspricht. So kann die Rolle flexibel an verschiedene Systeme angepasst und im selben Aufruf genutzt werden.

### 16.3.9 Aufgabenteil



## Aufgabenteil – Variablen in Rollen

- 1 Verfassen Sie eine Variablendatei für jede Distributionsfamilie der Umgebung im `vars`-Verzeichnis der Rolle `apache2_role`. Deklarieren Sie die Variable `apache2_package` mit dem entsprechenden Paketnamen des Apache2-Webservers.
- 2 Fügen Sie die Variablendateien in die Rolle `apache2_role` ein. Fügen Sie die Dateien so ein, dass abhängig von der Distributionsfamilie des Zielsystems die richtigen Variablen aufgerufen werden.
- 3 Ersetzen Sie den Servicenamen im Handler `restart webserver` in der `handlers/main.yml` durch die Variable `apache2_package`.

1. Verfassen Sie eine Variablendatei für jede Distributionsfamilie der Umgebung im `vars`-Verzeichnis der Rolle `apache2_role`. Deklarieren Sie die Variable `apache2_package` mit dem entsprechenden Paketnamen des Apache2-Webservers.
2. Fügen Sie die Variablendateien in die Rolle `apache2_role` ein. Fügen Sie die Dateien so ein, dass abhängig von der Distributionsfamilie des Zielsystems die richtigen Variablen aufgerufen werden.
3. Ersetzen Sie den Servicenamen im Handler `restart webserver` in der `handlers/main.yml` durch die Variable `apache2_package`.

## Musterlösung

1. Verfassen Sie eine Variablen datei für jede Distributionsfamilie der Umgebung im `vars`-Verzeichnis der Rolle `apache2_role`. Deklarieren Sie die Variable `apache2_package` mit dem entsprechenden Paketnamen des Apache2-Webservers:

Öffnen Sie einen Editor und erstellen Sie die folgenden Dateien:

Für Red Hat-Systeme `vars/RedHat.yml` mit dem Inhalt:

```
---
```

```
apache2_package: httpd
```

Für SUSE-Systeme `vars/Suse.yml` mit dem Inhalt:

```
---
```

```
apache2_package: apache2
```

Für Debian-Systeme `vars/Debian.yml` mit dem Inhalt:

```
---
```

```
apache2_package: apache2
```

2. Fügen Sie die Variablen dateien in die Rolle `apache2_role` ein. Fügen Sie die Dateien so ein, dass abhängig von der Distributionsfamilie des Zielsystems die richtigen Variablen aufgerufen werden:

Fügen Sie in der `tasks/main.yml` mit `include_vars`: die Variable für die Dateien in `vars/` ein:

```
---
```

```
- include_vars: "{{ ansible_os_family }}.yml"
- name: ensure that the "{{ apache2_package_state }}" version of
    ↪ the apache2 package is there
  ansible.builtin.package:
    name: "{{ apache2_package }}"
    state: "{{ apache2_package_state }}"
  notify:
    - restart webserver

- name: ensure that the service is up and running for the first
    ↪ time
  ansible.builtin.service:
    name: "{{ apache2_package }}"
    state: started
```

3. Ersetzen Sie den Servicenamen im Handlers `restart webserver` in der `handlers/main.yml` durch die Variable `apache2_package`:

Editieren Sie den Eintrag für den Servicenamen in der `main.yml` aus dem Handlers-Verzeichnis der Rolle `apache2_role`:

```
/root/code/roles/apache2_role/handlers/main.yml:  
---  
- name: restart webserver  
  ansible.builtin.service:  
    name: "{{ apache2_package }}"  
    state: restarted
```

Nun wird abhängig von der Distributionsfamilie immer das richtige Paket installiert und vom Handler neu gestartet, sofern sich der Zustand verändert hat.

### 16.3.10 Jinja2-Syntax



## Jinja2-Syntax

In Ansible zuständig für:

- Zugriff auf Variablen
- Erstellen von Vorlagen (templates)

### Zugriff auf Variablen

```
{{ variable }}  
{{ liste[0] }}  
{{ dictionary['key'] }}  
{{ dictionary.key }}
```

### Kommentare, die nicht im Zieldokument erscheinen

```
{# Kommentar #}
```

Bei Jinja2 handelt es sich um eine Templating-Sprache für Python. Jinja2 hat die Funktion, mit sogenannten *Templates* oder Mustern Vorlagen zu erschaffen, in denen bei der Verarbeitung Platzhalter durch Werte ersetzt werden. Solche Platzhalter können auf Variablen, Listen und *dictionaries* referieren und diese dynamisch einbinden. In Ansible wird Jinja2 z. B. dafür verwendet, Variablen aufzurufen, die in Kontextdateien in der YAML-Syntax deklariert wurden.

### Jinja2-Variablen in Playbooks

In Playbooks können Variablen oder auch Ansible Facts mit der Jinja2-Syntax referenziert werden. Die Variablen werden im `vars`-Dictionary deklariert und können dann mit geschweiften Klammern in Anführungszeichen ("`{ { } }`") referenziert werden:

```
---  
- name: a playbook with a variable  
  hosts: suse2  
  vars:  
    testvar: 'this is a test'  
  tasks:
```

```
- name: a task to reference the testvar
  ansible.builtin.debug:
    msg: "{{ testvar }}"
```

Die unter `vars:` deklarierte Variable `testvar` wird im `debug`-Modul im ersten Task mit der Option `msg:` in der Jinja2-Syntax ("`{{ testvar }}`") referenziert. Beim Ausführen des Tasks wird der Wert der Variablen ausgegeben:

```
# ansible-playbook -i inventory jinja2_testvar1.yml

PLAY [a playbook with a variable] **

TASK [Gathering Facts] **
ok: [suse2]

TASK [a task to reference the testvar] **
ok: [suse2] => {
    "msg": "this is a test"
}

PLAY RECAP **
suse2 : ok=2      changed=0      unreachable=0
      ↪ failed=0
```

Ansible Facts können auf dieselbe Art aufgerufen werden, wie deklarierte Variablen referenziert werden können:

```
---
- name: a playbook with a variable
  hosts: suse2
  tasks:
    - name: a task to reference an ansible fact
      ansible.builtin.debug:
        msg: "{{ ansible_hostname }}"
```

Nun gibt das `debug`-Modul den Wert des Ansible Facts `ansible_hostname` aus:

```
# ansible-playbook -i inventory jinja2_ansible_fact1.yml

PLAY [a playbook with a variable] **

TASK [Gathering Facts] **
ok: [suse2]

TASK [a task to reference an ansible fact] **
ok: [suse2] => {
    "msg": "suse2"
}

PLAY RECAP **
suse2 : ok=2      changed=0      unreachable=0
      ↪ failed=0
```



# Jinja2-Syntax – Zugriff auf Listen und Variablen

Mit Jinja2:

- einzelne Listenpunkte referenzieren
- Keys aus verschachtelten Dictionaries referenzieren

## Zugriff auf vierten Listenpunkt

```
{% liste[3] %}
```

## Zugriff auf einen verschachtelten Key

```
testdict:
  detail: 'is green'
  subdict:
    reference: 'here it is'

{{ testdict['subdict']['reference'] }}
```

Der Zugriff auf Listen oder auf *dictionaries* geschieht in derselben Schreibweise mit geschweiften Klammern.

Für eine Liste:

```
# cat host_vars/suse2.yml
liste:
  - 'first point'
  - 'second point'
  - 'third point'
```

Im Playbook wird nun die Liste in der `loop`-Schleife in der selben Art referenziert wie eine einfache Variable:

```
---
- name: a playbook with a loop through a list
  hosts: suse2
  tasks:
    - name: a task to reference a list with a loop
      ansible.builtin.debug:
        msg: "{{ item }}"
      loop: "{{ liste }}"
```

**Das Resultat:**

```
# ansible-playbook -i inventory jinja2_ansible_list1.yml

PLAY [a playbook with a loop through a list] **

TASK [Gathering Facts] **
ok: [suse2]

TASK [a task to reference a list with a loop] **
ok: [suse2] => (item=first point) => {
    "msg": "first point"
}
ok: [suse2] => (item=second point) => {
    "msg": "second point"
}
ok: [suse2] => (item=third point) => {
    "msg": "third point"
}

PLAY RECAP **
suse2 : ok=2     changed=0     unreachable=0
      ↳ failed=0
```

Soll nicht die gesamte Liste, sondern lediglich ein einzelner Eintrag referenziert werden, so wird die Position des Eintrags – nach dem Feldindex gezählt – in eckigen Klammern angegeben:

```
---
- name: a playbook with a reference to a value in list
  hosts: suse2
  tasks:
    - name: a task to reference values in a list
      ansible.builtin.debug:
        msg: "{{ liste[1] }}"
```

**Das debug-Modul gibt nun den zweiten Eintrag der Liste liste aus:**

```
# ansible-playbook -i inventory jinja2_ansible_list2.yml

PLAY [a playbook with a reference to a value in list] **

TASK [Gathering Facts] **
ok: [suse2]

TASK [a task to reference values in a list] **
ok: [suse2] => {
    "msg": "second point"
}

PLAY RECAP **
suse2 : ok=2     changed=0     unreachable=0
      ↳ failed=0
```

## 16 Ansible Playbooks und Roles

Um auf mehrere Punkte einer Liste zuzugreifen, müssen die zusätzlichen Listenzugriffe separat in der Jinja2-Syntax angegeben werden. Hierfür wird der Aufruf der `msg`-Option ergänzt:

```
---
- name: a playbook with a reference to a value in list
  hosts: suse2
  tasks:
    - name: a task to reference values in a list
      ansible.builtin.debug:
        msg: "{{ liste[1] }} {{ liste[2] }}"
```

Nun werden zwei Punkte ausgegeben:

```
# ansible-playbook -i /root/code/inventory
  ↪ /root/code/jinja2_ansible_list3.yml

PLAY [a playbook with a reference to a value in list] **

TASK [Gathering Facts] **
ok: [suse2]

TASK [a task to reference values in a list] **
ok: [suse2] => {
    "msg": "second point third point"
}

PLAY RECAP **
suse2 : ok=2     changed=0     unreachable=0
  ↪ failed=0
```

Neben dem Zugriff auf Variablen und Listen kann mit der Jinja2-Syntax auch auf *ictionaries* und einzelne Einträge daraus zugegriffen werden.

```
# cat host_vars/suse2.yml

testdict:
  name: 'first name'
  detail: 'is green'
  subdict:
    reference: 'here it is'
```

Damit ein bestimmter Eintrag aus einem Dictionary referenziert werden kann, wird dem Aufruf des Dictionary in geschweiften Klammern noch der Name (Key) des Eintrags in eckigen Klammern angehängt. Im Folgenden wird der Eintrag `detail` des Dictionary `testdict` referenziert:

```
---
- name: a playbook with a reference to a dictionary
  hosts: suse2
  tasks:
    - name: a task to reference values in a dictionary
      ansible.builtin.debug:
```

```
msg: "{{ testdict['detail'] }}"
```

In der Ausgabe wird der Wert des Eintrags `detail` ausgegeben:

```
# ansible-playbook -i inventory jinja2_ansible_dict1.yml

PLAY [a playbook with a reference to a dictionary] **

TASK [Gathering Facts] **
ok: [suse2]

TASK [a task to reference values in a dictionary] **
ok: [suse2] => {
    "msg": "is green"
}

PLAY RECAP **
suse2 : ok=2     changed=0     unreachable=0
      ↳ failed=0
```

Um auf die tiefer liegenden *dictionaries* zuzugreifen, müssen die weiteren Einträge ebenfalls in eckigen Klammern angehängt werden:

```
# cat host_vars/suse2.yml

testdict:
  name: 'first name'
  detail: 'is green'
  subdict:
    reference: 'here it is', Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de
```

Nun soll der Eintrag `reference` ausgegeben werden:

```
---
- name: a playbook with a reference to a dictionary
  hosts: suse2
  tasks:
    - name: a task to reference values in a dictionary
      ansible.builtin.debug:
        msg: "{{ testdict['subdict']['reference'] }}"
```

Bei Ausführung des Playbook wird der Wert des Eintrags `reference` ausgegeben:

```
# ansible-playbook -i inventory jinja2_ansible_dict2.yml

PLAY [a playbook with a reference to a dictionary] **

TASK [Gathering Facts] **
ok: [suse2]

TASK [a task to reference values in a dictionary] **
ok: [suse2] => {
    "msg": "here it is"
}
```

## 16 Ansible Playbooks und Roles

```
PLAY RECAP **  
suse2 : ok=2     changed=0    unreachable=0  
      ↳ failed=0
```

Mit dieser Art der Referenzierung können *dictionaries* in beliebiger Größe und Tiefe referenziert werden.

### 16.3.11 Aufgabenteil



## Aufgabenteil – Jinja2-Syntax

- ① Deklarieren Sie eine Liste mit sechs Einträgen als Variable für das Zielsystem centos2. Schreiben Sie ein Playbook, das einen Task enthält, der den dritten und den fünften Eintrag mit dem ansible.builtin.debug-Modul ausgibt.
- ② Schreiben Sie ein Playbook mit einem Task, in dem mit dem ansible.builtin.debug-Modul die IP-Adresse des Zielsystems centos2 aus dem Dictionary des Ansible Facts ipv4 ausgegeben wird.

1. Deklarieren Sie eine Liste mit sechs Einträgen als Variable für das Zielsystem centos2. Schreiben Sie ein Playbook, das einen Task enthält, der den dritten und den fünften Eintrag mit dem ansible.builtin.debug-Modul ausgibt.
2. Schreiben Sie ein Playbook mit einem Task, in dem mit dem ansible.builtin.debug-Modul die IP-Adresse des Zielsystems centos2 aus dem Dictionary des Ansible Facts ipv4 ausgegeben wird.

## Musterlösung

1. Deklarieren Sie eine Liste mit sechs Einträgen als Variable für das Zielsystem `centos2`. Schreiben Sie ein Playbook, das einen Task enthält, der den dritten und den fünften Eintrag mit dem `ansible.builtin.debug`-Modul ausgibt:

Legen Sie die Datei `/root/code/jinja2_exercise_list.yml` für das Playbook an.

Erstellen Sie eine Liste mit sechs Punkten, z. B. als Variable direkt im Playbook. Weitere mögliche Orte zur Deklaration der Liste wären die `host_vars-` oder `group_vars`-Verzeichnisse, die Datei `defaults/main.yml` einer Rolle oder eine Datei im `vars/-`-Verzeichnis einer Rolle.

Fügen Sie anschließend dem Playbook einen Task hinzu, in dem die Liste mit dem `ansible.builtin.debug`-Modul und der Option `msg` referenziert wird. Beachten Sie die Angabe der Listenpunkte nach dem Feldindex:

```
---
- name: a playbook to use jinja2 syntax
  hosts: centos2
  vars:
    exercise_list:
      - 'zero'
      - 'one'
      - 'two'
      - 'three'
      - 'four'
      - 'five'
  tasks:
    - name: a task with a debug modul to sort things out
      ansible.builtin.debug:
        msg: "{{ exercise_list[3] }} {{ exercise_list[5] }}"
```

Bei Ausführung gibt das Playbook die Werte der Listenpunkte drei und fünf aus:

```
# ansible-playbook -i /root/code/inventory
  ↪ /root/code/jinja2_exercise_list.yml

PLAY [a playbook to use jinja2 syntax] **

TASK [Gathering Facts] **
ok: [centos2]

TASK [a task with a debug modul to sort things out] **
ok: [centos2] => {
      "msg": "three five"
}

PLAY RECAP **
centos2 : ok=2      changed=0      unreachable=0
         failed=0
```

- 2. Schreiben Sie ein Playbook mit einem Task, in dem mit dem `ansible.builtin.debug`-Modul die IP-Adresse des Zielsystems `centos2` aus dem Dictionary des Ansible Facts `ipv4` ausgegeben wird:**

Legen Sie ein weiteres Playbook an, z.B.

```
/root/code/jinja2_exercise_facts.yml
```

Geben Sie mit der Option `msg` des `ansible.builtin.debug`-Moduls das richtige Dictionary an:

```
---
- name: a playbook to use jinja2 syntax
  hosts: centos2
  tasks:
    - name: a task with a debug modul to sort things out
      ansible.builtin.debug:
        msg: "{{ ansible_eth0['ipv4']['address'] }}"
```

Der Task in dem Playbook gibt nun die IP-Adresse aus:

```
# ansible-playbook -i /root/code/inventory
  ↪ /root/code/jinja2_exercise_facts.yml

PLAY [a playbook to use jinja2 syntax] **

TASK [Gathering Facts] **
ok: [centos2]

TASK [a task with a debug modul to sort things out] **
ok: [centos2] => {
      "msg": "172.20.1.102"
}

PLAY RECAP **
centos2 : ok=2    changed=0    unreachable=0
  ↪ failed=0
```

### 16.3.12 Jinja2-Syntax: Kontrollstrukturen



## Kontrollstrukturen in Jinja2 1/2

- mit Konditionalen und Schleifen den Wert von Variablen festlegen
- relevant in Verbindung mit dem `ansible.builtin.template`-Modul

### Kontrollstrukturen – Schleife

```
{% for i in x %}
{{ i }}
{% endfor %}
```

### Beispiel einer Schleife in einer .j2-Template-Datei

```
{% for x in ansible_interfaces %}
iface: {{ x }}
{% endfor %}
```

Mit der Jinja2-Syntax lassen sich Kontrollstrukturen formulieren, die anhand bestimmter Prozesszustände verschiedene Bedingungen prüfen und danach Befehle ausführen können. Zu diesen Kontrollmechanismen gehören zum einen Schleifen und zum anderen wenn-dann-Überprüfungen, von denen der Inhalt der zu erstellenden Templates abhängig gemacht werden kann.

### Schleifen in Jinja2

Neben dem Ausgeben von Werten aus Variablen kann mit Jinja2 auch in Schleifen über Listen und *dictionaries* iteriert werden. Die Syntax sieht dabei wie folgt aus

```
{% for i in x %}
{{ i }}
{% endfor %}
```

wobei `i` das Iterationsobjekt und `x` das Listenobjekt bezeichnet.

Um eine Schleife zu erstellen, legen Sie eine Datei an, welche die Jinja2-Syntax enthält. Jinja2 ist auf keine bestimmte Dateiendung festgelegt, jedoch bietet es sich an, reine Jinja2-

Dateien mit einem Suffix `.j2` zu versehen. Eine Datei mit der Schleife könnte folgenden Inhalt haben:

```
# cat /root/code/jinja2_examples/loop.j2

{% for i in testlist %}
{{ i }}
{% endfor %}
```

Die Liste, die unter der Variable „`testlist`“ aufgerufen wird, kann in Ansible entweder direkt im Playbook, einer Datei in den `host_vars-` oder `group_vars`-Verzeichnissen, der `defaults/main.yml` einer Rolle oder dem `vars/-Verzeichnis` einer Rolle angelegt werden. Im Folgenden ist die Liste `testlist` als Hostvariable für das Zielsystem `ubuntul` deklariert:

```
# cat host_vars/ubuntul.yml

---
testlist:
- test0
- test1
- test2
```

Die Datei mit der Jinja2-Syntax kann nun über das `template`-Modul in ein Playbook eingebunden werden:

```
---
- name: a playbook to test some jinja2
hosts: ubuntul
tasks:
- name: a templating task to check out the jinja2
  ansible.builtin.template:
    src: /root/code/jinja2_examples/loop.j2
    dest: /root/test/list.txt
```

Wird das Playbook nun ausgeführt:

```
# ansible-playbook -i inventory jinja2_for_loopl.yml

PLAY [a playbook to test some jinja2] **

TASK [Gathering Facts] **
ok: [ubuntul]

TASK [a templating task to check out the jinja2] **
changed: [ubuntul]

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

so erstellt der Task mit dem `template`-Modul unter dem in der Option `dest` angegebenen Pfad auf dem Zielsystem `ubuntul` die Datei `list.txt` mit dem Inhalt der in der `host_vars/ubuntul.yml` deklarierten Liste:

## 16 Ansible Playbooks und Roles

```
[root@ubuntu1 ~]# cat /root/test/list.txt
test0
test1
test2
```

Das `template`-Modul wertet die Jinja2-Syntax aus und iteriert mit der dort formulierten Schleife durch die einzelnen Listenpunkte der deklarierten Liste.

Mit der `for`-Schleife in Jinja2 lassen sich neben benutzerdeklarierten Listen auch Ansible Facts auswerten. Dazu kann durch die Facts iteriert werden, um z. B. eine Liste mit allen eingebundenen Laufwerken zu erzeugen:

```
# cat /root/code/jinja2_examples/loop_ansible_facts.j2
```

```
{% for x in ansible_interfaces %}
iface: {{ x }}
{% endfor %}
```

In diesem Fall iteriert die Schleife durch das Dictionary des Ansible Facts `ansible_interfaces`. Die Jinja2-Datei wird wieder in einem Playbook mit dem `template`-Modul aufgerufen:

```
---
- name: a playbook to test some jinja2
  hosts: ubuntu1
  tasks:
    - name: a templating task to check out the jinja2
      ansible.builtin.template:
        src: /root/code/jinja2_examples/loop_ansible_facts.j2
        dest: /root/test/facts.txt
```

Ausführen des Playbooks:

```
# ansible-playbook -i /root/code/inventory
  ↪ /root/code/jinja2_for_loop1.yml

PLAY [a playbook to test some jinja2] **

TASK [Gathering Facts] **
ok: [ubuntu1]

TASK [a templating task to check out the jinja2] **
changed: [ubuntu1]

PLAY RECAP **
ubuntu1 : ok=2      changed=1      unreachable=0
  ↪ failed=0
```

Als Ergebnis erzeugt der Task auf dem Zielsystem `ubuntu1` die Textdatei mit dem Inhalt:

```
[root@ubuntu1 ~]# cat /root/test/facts.txt
iface: lo
iface: eth0
```



## Kontrollstrukturen in Jinja2 2/2

### Kontrollstrukturen – If-Else

```
{% if path == '/root/test' %}
    {% set value = value1 %}
{% else %}
    {% set value = value2 %}
{% endif %}
```

### Eine Schleife in einer If-Else Bedingung

```
{% if path == '/root/test' %}
{% for i in testlist %}
    {{ i }}
{% endfor %}
{% else %}
    {{ testlist[2] }}
{% endif %}
```

Jinja2 verfügt auch über die Möglichkeit, innerhalb von Templates wenn-dann-Überprüfungen durchzuführen und abhängig von deren Ausgang verschiedene Befehle auszuführen. Die generelle Syntax dieser Abfragen gestaltet sich wie folgt:

```
{% if %}
{%
  elif %}
{%
  else %}
{%
  endif %}
```

Die If-Else-Konstruktion kann beispielsweise dazu genutzt werden, bestimmte Variablen in Abhängigkeit einer Bedingung zu deklarieren. Hierfür muss zunächst eine Jinja2-Datei erstellt werden, in welcher die Bedingung ausformuliert ist.

Im Beispiel wird die Bedingung in der Datei `/root/code/jinja2_if_else.j2` abgelegt:

```
{% if path == '/root/test' %}
    {% set value = value1 %}
{% else %}
    {% set value = value2 %}
{% endif %}
{{ value }}
```

## 16 Ansible Playbooks und Roles

Hier wird festgelegt, dass wenn die Variable path dem Pfad /root/test entspricht, für die Variable value der Wert der Variablen value1 gesetzt wird. Hat die Variable path einen anderen Wert, wird für die Variable value der Wert der Variablen value2 gesetzt. Am Ende wird der Wert der Variablen value in die vom template-Modul erzeugte Datei eingetragen. Die Variablen, zwischen denen die Bedingung entscheiden soll, können direkt im Playbook, einer Datei in den host\_vars- oder group\_vars-Verzeichnissen, der defaults/main.yml einer Rolle oder dem vars-/Verzeichnis einer Rolle deklariert werden. Im Folgenden werden sie direkt in dem Playbook deklariert, welches auch den Task mit dem template-Modul enthält:

```
# cat /root/code/jinja2_ansible_if_else.yml

---
- name: a playbook to test some jinja2
  hosts: ubuntu1
  vars:
    value1: 'This is the variable Value1'
    value2: 'This is the variable Value2'
    path: '/root/test'
  tasks:
    - name: a templating task to check out the jinja2
      ansible.builtin.template:
        src: /root/code/jinja2_examples/jinja2_if_else.j2
        dest: /root/test/if.txt
```

Wird das Playbook ausgeführt:

```
# ansible-playbook -i inventory jinja2_ansible_if_else.yml

PLAY [a playbook to test some jinja2] **

TASK [Gathering Facts] **
ok: [ubuntu1]

TASK [a templating task to check out the jinja2] **
changed: [ubuntu1]

PLAY RECAP **
ubuntu1 : ok=2      changed=1      unreachable=0
          ↵ failed=0
```

so erstellt es eine Datei /root/test/if.txt auf dem Zielsystem, in welcher der Wert der Variablen value eingetragen ist. Je nachdem, ob die Bedingung erfüllt wird, enthält diese den Wert der Variablen value1 oder value2. Da in dem Playbook die Variable path so deklariert ist, dass die Bedingung zutrifft, enthält die Datei if.txt den Wert der Variablen value1:

```
[root@ubuntu1 ~]# cat /root/test/if.txt
This is the variable Value1
```



# Jinja2 und Template-Modul

## Inhalt der jinja2\_ansible\_if\_elseif.j2-Datei:

```
{% if path == '/root/test' %}
    {% set value = value1 %}
{% elif path == '/root/test/foo' %}
    {% set value = value2 %}
{% elif path == '/root/test/bar' %}
    {% set value = value3 %}
{% else %}
    {% set value = value4 %}
{% endif %}
{{ value }}
```

## Task mit dem ansible.builtin.template-Modul:

```
- name: a templating task to check out the jinja2
  ansible.builtin.template:
    src: /root/code/jinja2_examples/jinja2_ansible_if_elseif.j2
    dest: /root/test/if.txt
```

Neben der einfachen If-Else-Abfrage kann die Bedingung auch um weitere Möglichkeiten erweitert werden. Hierfür nutzt Jinja2 die `elif`-Bedingung, mit der alternative Zustände beschrieben werden können.

Im folgenden Beispiel wurde das Jinja2-Template um zwei `elif`-Bedingungen erweitert:

```
{% if path == '/root/test' %}
    {% set value = value1 %}
{% elif path == '/root/test/foo' %}
    {% set value = value2 %}
{% elif path == '/root/test/bar' %}
    {% set value = value3 %}
{% else %}
    {% set value = value4 %}
{% endif %}
{{ value }}
```

Damit die alternativen Bedingungen zutreffen können, müssen die entsprechenden Variablen `value1` bis `value4` deklariert werden:

```
---
- name: a playbook to test some jinja2
  hosts: ubuntu1
  vars:
```

## 16 Ansible Playbooks und Roles

```
value1: 'This is the variable Value1'
value2: 'This is the variable Value2'
value3: 'This is the variable Value3'
value4: 'This is the variable Value4'
path:  '/root/test/'

tasks:

  - name: a templating task to check out the jinja2
    ansible.builtin.template:
      src: /root/code/jinja2_examples/jinja2_ansible_if_elseif.j2
      dest: /root/test/if.txt
```

Nun wird, je nachdem welcher Wert in der Variablen path deklariert wird, ein anderer Wert für die Variable value gesetzt. Entspricht path /root/test, so wird value1 gesetzt. Es wird value2 gesetzt, wenn path /root/test/foo entspricht. Oder es wird value3 gesetzt, wenn path /root/test/bar entspricht. Wenn path allerdings keinem der alternativen Werte entspricht, wird in der Variable value der Wert von value4 deklariert. Schließlich wird der Wert von value in die im template-Modul angegebene Datei geschrieben.

### Else-If-Konditionen mit Schleifen kombinieren

In der Jinja2-Syntax ist es möglich, die Konditionalen mit Schleifen zu kombinieren. Es kann zum Beispiel eine Schleife ausgeführt werden, wenn eine Bedingung zutrifft. Formal sieht dies so aus:

```
{% if   %}
{%- for %}

{% endfor %}
{% else  %}

{% endif  %}
```

Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de

Im folgenden Beispiel wird eine Jinja2-Datei von einem Task in einem Playbook mit dem template-Modul aufgerufen:

```
---
- name: a playbook to test some jinja2
  hosts: ubuntu1
  vars:
    testlist:
      - test2
      - test5
      - test0
    value1: 'This is the variable Value1'
    value2: 'This is the variable Value2'
    path:  '/root/test'
  tasks:
    - name: a templating task to check out the jinja2
```

```
ansible.builtin.template:  
  src: /root/code/jinja2_examples/jinja2_ansible_if_elseif_for.j2  
  dest: /root/test/if.txt
```

In der Jinja2-Datei sind die Bedingungen definiert, unter denen eine Schleife ausgeführt wird:

```
{% if path == '/root/test' %}  
{% for i in testlist %}  
  {{ i }}  
{% endfor %}  
{% else %}  
  {{ testlist[2] }}  
{% endif %}
```

Trifft die Bedingung zu, dass path den Wert /root/test hat, wird eine For-Schleife ausgeführt, die über die Liste testlist iteriert und dabei alle Punkte der Liste in die Datei /root/test/if.txt auf dem Zielsystem schreibt. Wenn die Variable path einen anderen Wert enthält, so wird lediglich der dritte Listenpunkt in if.txt geschrieben.



### 16.3.13 Aufgabenteil



## Aufgabenteil – Jinja2 Kontrollstrukturen

- ① Deklarieren Sie eine Liste `jinjalist` mit fünf Punkten in den `group_vars` für die Gruppe `webserver`. Schreiben Sie eine Jinja2-Datei, in der eine For-Schleife über diese Liste iteriert. Verfassen Sie ein Playbook, in dem mit dem `ansible.builtin.template`-Modul eine Datei `jinjalist.txt` im Verzeichnis `/root` auf den Systemen der Gruppe `webserver` erzeugt wird.
- ② Schreiben Sie eine Jinja2-Datei mit einem If-Else-Konditional, das je nach `ansible_os_family` einen anderen Satz erzeugt. Schreiben Sie ein Playbook, in dem Sie die Datei mit dem `template`-Modul auswerten und auf allen Zielsystemen eine Datei erstellen lassen.

1. Deklarieren Sie eine Liste `jinjalist` mit fünf Punkten in den `group_vars` für die Gruppe `webserver`. Schreiben Sie eine Jinja2-Datei, in der eine For-Schleife über diese Liste iteriert. Verfassen Sie ein Playbook, in dem mit dem `template`-Modul eine Datei `jinjalist.txt` im Verzeichnis `/root` auf den Systemen der Gruppe `webserver` erzeugt wird.
2. Schreiben Sie eine Jinja2-Datei mit einem If-Else-Konditional, das je nach `ansible_os_family` einen anderen Satz erzeugt. Schreiben Sie ein Playbook, in dem Sie die Datei mit dem `ansible.builtin.template`-Modul auswerten und auf allen Zielsystemen eine Datei erstellen lassen.

## Musterlösung

1. Deklarieren Sie eine Liste `jinjalist` mit fünf Punkten in den `group_vars` für die Gruppe `webserver`. Schreiben Sie eine Jinja2-Datei, in der eine For-Schleife über diese Liste iteriert. Verfassen Sie ein Playbook, in dem mit dem `template`-Modul eine Datei `jinjalist.txt` im Verzeichnis `/root` auf den Systemen der Gruppe `webserver` erzeugt wird:

Fügen Sie in die Datei `/root/code/group_vars/webserver.yml` die Liste `jinjalist` mit den fünf Punkten ein:

```
---
jinjalist:
  - "the first point is the best"
  - "this could be a value"
  - "remeber the field index"
  - "so this i 3, right?"
  - "4-5"
```

Erstellen Sie nun die Datei

`/root/code/jinja2_examples/ansible_jinjalist_exercise.j2` und fügen Sie hier die For-Schleife in der Jinja2-Syntax ein:

```
{% for i in jinjalist %}
{{ i }}
{% endfor %}
```

Schreiben Sie nun das Playbook mit dem `template`-Modul in `/root/code/jinja2_exercise_for.yml`. Geben Sie bei `hosts:` die Gruppe `webserver` an und fügen Sie einen Task mit dem `ansible.builtin.template`-Modul ein:

```
---
- name: a playbook to run a template modul
  hosts: webserver
  tasks:
    - name: the template task
      ansible.builtin.template:
        src:
          ↪ /root/code/jinja2_examples/ansible_jinjalist_exercise.j2
        dest: /root/jinjalist.txt
```

Bei Ausführung des Playbooks erstellt der Task jeweils eine Datei `jinjalist.txt` auf jedem System:

```
# ansible-playbook -i /root/code/inventory
  ↪ /root/code/jinja2_exercise_for.yml
```

```
PLAY [a playbook to run a template modul] **

TASK [Gathering Facts] **
ok: [ubuntu1]
ok: [centos2]
```

```

TASK [the template task] **
changed: [ubuntul]
changed: [centos2]

PLAY RECAP **
centos2 : ok=2    changed=1    unreachable=0
         ↪ failed=0
ubuntul : ok=2    changed=1    unreachable=0
         ↪ failed=0

```

Auf den Systemen der Gruppe webserver:

```

root@centos2 ~]# cat /root/jinjalist.txt
the first point is the best
this could be a value
remeber the field index
so this is 3, right?
4-5

```

```

root@ubuntul ~]# cat jinjalist.txt
the first point is the best
this could be a value
remeber the field index
so this is 3, right?
4-5

```

2. Schreiben Sie eine Jinja2-Datei mit einem If-Else-Konditional, das je nach `ansible_os_family` einen anderen Satz erzeugt. Schreiben Sie ein Playbook, in dem Sie die Datei mit dem `ansible.builtin.template`-Modul auswerten und auf allen Zielsystemen eine Datei erstellen lassen:

Legen Sie eine Datei an, z. B.

```
/root/code/jinja2_examples/jinja2_ansible_if_elseif_exercise_os.j2
```

Erstellen Sie ein If-Else Konditional, in welchem je nach `ansible_os_family` ein anderer Satz ausgewählt wird:

```

{% if ansible_os_family == "Debian" %}
Debian and GNU are just for you!
{% elif ansible_os_family == "RedHat" %}
Look at my faboulus Fedora!
{% elif ansible_os_family == "Suse" %}
Do you see the CHAMELEON?
{% else %}
What is this??
{% endif %}

```

Verfassen Sie nun das Playbook mit dem `template`-Modul, z. B. in

`/root/code/jinja2_ansible_if_elseif_exercise_os.yml`. Tragen Sie alle Zielsysteme ein und rufen Sie die Jinja2-Datei auf:

```

---
- name: a playbook to test some jinja2
  hosts: all

```

## 16 Ansible Playbooks und Roles

tasks:

```
- name: a templating task to check out the jinja2
  ansible.builtin.template:
    src: /root/code/jinja2_examples/
      ↪ jinja2_ansible_if_elseif_exercise_os.j2
    dest: /root/test/if_exercise.txt
```

Bei Ausführung des Playbooks:

```
ansible-playbook -i /root/code/inventory
  ↪ /root/code/jinja2_ansible_if_elseif_exercise_os.yml
```

```
PLAY [a playbook to test some jinja2] **
```

```
TASK [Gathering Facts] **
ok: [ubuntu1]
ok: [ubuntu2]
ok: [suse2]
ok: [suse1]
ok: [centos2]
ok: [centos1]
```

```
TASK [a templating task to check out the jinja2] **
changed: [ubuntu1]
changed: [centos2]
changed: [centos1]
changed: [suse1]
changed: [suse2]
changed: [ubuntu2]
```

```
PLAY RECAP **
centos1          : ok=2    changed=1    unreachable=0
  ↪ failed=0
centos2          : ok=2    changed=1    unreachable=0
  ↪ failed=0
suse1            : ok=2    changed=1    unreachable=0
  ↪ failed=0
suse2            : ok=2    changed=1    unreachable=0
  ↪ failed=0
ubuntul          : ok=2    changed=1    unreachable=0
  ↪ failed=0
ubuntu2          : ok=2    changed=1    unreachable=0
  ↪ failed=0
```

Ob die Sätze jeweils in die Datei `if_exercise.txt` geschrieben wurden, lässt sich mit einem Playbook überprüfen:

```
---
- name: a playbook to read some files
  hosts: all
  tasks:
    - name: command with a registered result
      ansible.builtin.command: cat /root/test/if_exercise.txt
```

```

    register: cat_register

- name: the debug to display the cat
  ansible.builtin.debug:
    msg: "{{ cat_register['stdout'] }}"

```

Der erste Task gibt den Inhalt der Datei `if_exercise.txt` für jedes System mit dem Befehl `cat` aus und registriert ihn in der Variablen `cat_register`. Der zweite Task gibt die Variable `cat_register` mit dem `debug`-Modul aus:

```

# ansible-playbook -i /root/code/inventory cat.yml

PLAY [a playbook to read some files] **

TASK [Gathering Facts] **
ok: [ubuntu1]
ok: [ubuntu2]
ok: [suse2]
ok: [suse1]
ok: [centos2]
ok: [centos1]

TASK [command with a registered result] **
changed: [ubuntu1]
changed: [ubuntu2]
changed: [centos2]
changed: [centos1]
changed: [suse2]
changed: [suse1]

TASK [the debug to display the cat] **
ok: [centos1] => {
    "msg": "Look at my faboulus Fedora!"
}
ok: [centos2] => {
    "msg": "Look at my faboulus Fedora!"
}
ok: [suse1] => {
    "msg": "Do you see the CHAMELEON?"
}
ok: [suse2] => {
    "msg": "Do you see the CHAMELEON?"
}
ok: [ubuntu1] => {
    "msg": "Debian and GNU are just for you!"
}
ok: [ubuntu2] => {
    "msg": "Debian and GNU are just for you!"
}

PLAY RECAP **
centos1                  : ok=3      changed=1      unreachable=0
                           ↳      failed=0

```

## 16 Ansible Playbooks und Roles

```
centos2          : ok=3      changed=1      unreachable=0
  ↳ failed=0
suse1           : ok=3      changed=1      unreachable=0
  ↳ failed=0
suse2           : ok=3      changed=1      unreachable=0
  ↳ failed=0
ubuntul         : ok=3      changed=1      unreachable=0
  ↳ failed=0
ubuntu2         : ok=3      changed=1      unreachable=0
  ↳ failed=0
```

Auf jedem System wurde eine Datei `if_exercise.txt` erstellt, in der jeweils ein Satz entsprechend des Facts `ansible_os_family` erzeugt wurde.

### 16.3.14 Jinja2-Syntax: Filter & Tests



## Jinja2-Syntax: Filter & Tests

- Filter: Werkzeuge zum Formatieren von Variablen
- z. B. `join(' ', )`, `capitalize`
- Werte werden mit Pipe (`|`) an Filter übergeben

Variablen, deren Werte an Filter übergeben werden:

```
Benutzer: {{ users | join(', ') }}  
Login erlaubt? {{ login_erlaubt | default('no') }}
```

Task in dem die Ausgabe mit Filtern formatiert wird:

```
- name: a task to filter some facts  
ansible.builtin.debug:  
  msg: "{{ ansible_mounts | map(attribute='mount') }}"
```

Neben den Kontrollmechanismen können in Jinja2 auch Filter sowie weitere Tests eingesetzt werden. Die Filter dienen maßgeblich dazu, Daten und Variablen zu formatieren. Tests haben die Aufgabe, weitere Bedingungen zu prüfen, um den Prozessverlauf automatisiert zu kontrollieren.

### Filter in Jinja2

Jinja2 verfügt über eine große Auswahl an Filtern, anhand derer Daten verarbeitet und formatiert werden können.

Die Filter werden mit dem Operator (`|`) auf den zu filternden Jinja2-Ausdruck angewendet. Das folgende Beispiel wendet den Filter `first` auf eine Liste mit vier Werten an. Der Filter filtert die Liste nach dem ersten Wert und gibt diesen aus:

```
{{ [1, 2, 3, 4] | first }}
```

Das Ergebnis ist der erste Wert der Liste:

1

## 16 Ansible Playbooks und Roles

Der Filter `length` gibt hingegen die Anzahl der Werte in der Liste wieder:

```
{{ [1, 2, 3, 4] | length }}
```

Das Ergebnis lautet also:

```
4
```

Filter können dabei auch kombiniert werden. Dabei wird der Ausgangswert durch den ersten Filters ausgewertet, und das Ergebnis dann an den folgenden Filter weitergegeben:

```
{{ [1, 2, 6, 3, 4] | reject('odd') | min }}
```

In diesem Fall verwirft der Filter `reject` alle ungerade Zahlenwerte aus der Liste und übergibt anschließend die übriggebliebenen, geraden Werte an den Filter `min`, der den kleinsten Wert ausgibt. Das Ergebnis ist dementsprechend:

```
2
```

### Jinja2-Filter in Ansible

Die von Jinja2 zur Verfügung gestellten Filter können auch in Ansible eingesetzt werden, um z. B. Ansible Facts oder Variablen zu filtern. Dafür werden die Filter einfach an den entsprechenden Stellen in die Jinja2-Syntax eingefügt:

```
---
- name:
  hosts: ubuntu1
  tasks:
    - name: a task to filter some facts
      ansible.builtin.debug:
        msg: "{{ ansible_mounts | map(attribute='mount') | list }}"
```

Bei Ausführung des Playbooks liefert das `ansible.builtin.debug`-Modul eine formatierte Liste, da der Filter `map` alle Objekte mit dem Wert `mount` aus der Liste des Ansible Fact `ansible_mounts` herausfiltert und mit dem Filter `list` wieder zu einer Liste formatiert:

```
# ansible-playbook -i inventory jinja2_filter_ansible.yml

PLAY [ubuntu1] **

TASK [Gathering Facts] **
ok: [ubuntu1]

TASK [a task to filter some facts] **
ok: [ubuntu1] => {
  "msg": [
    "/etc/resolv.conf",
    "/etc/hostname",
    "/etc/hosts",
```

```

        "/run/secrets"
    ]
}

PLAY RECAP **
ubuntu1 : ok=2      changed=0      unreachable=0
          ↏ failed=0

```

Mit dem Filter `join()` können Strings in die Ergebnisse eingefügt werden und diese somit formatieren:

```

---
- name: a playbook to use some filter in ansible
  hosts: centos1
  tasks:
    - name: a task to get some input
      ansible.builtin.command: cat /etc/filesystems
      register: command_result

    - name: a debug task to represent
      ansible.builtin.debug:
        msg: "{{ command_result['stdout_lines'] | join(',') }}"

```

Nun sind die Ergebnisse durch Kommata separiert:

```

# ansible-playbook -i inventory jinja2_filter_ansible_2.yml

PLAY [a playbook to use some filter in ansible] **

TASK [Gathering Facts] Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de
ok: [centos1]

TASK [a task to get some input] **
changed: [centos1]

TASK [a debug task to represent] **
ok: [centos1] => {
    "msg": "xfs,ext4,ext3,ext2,nodev proc,nodev
           ↏ devpts,iso9660,vfat,hfs,hfsplus,*"
}

PLAY RECAP **
centos1 : ok=3      changed=1      unreachable=0
          ↏ failed=0

```

Mit dem Filter `default()` vergeben Sie Standardwerte für bestimmte Variablen. Diese gelten, wenn die aufgerufene Variable nicht deklariert wurde:

```

- name: a playbook to use some filter in ansible
  hosts: centos1
  tasks:
    - name:
      ansible.builtin.debug:
        msg: "{{ remote_login | default('no') }}"

```

## 16 Ansible Playbooks und Roles

Wurde die Variable `remote_login` nicht deklariert, so wird an ihrer Stelle standardmäßig der Wert `no` gesetzt:

```
# ansible-playbook -i inventory jinja2_filter_ansible_2.yml

PLAY [a playbook to use some filter in ansible] **

TASK [Gathering Facts] **
ok: [centos1]

TASK [debug] **
ok: [centos1] => {
    "msg": "no"
}

PLAY RECAP **
centos1 : ok=2      changed=0      unreachable=0
          ↳ failed=0
```

Eine Übersicht über alle verfügbaren Jinja2-Filter finden Sie unter:

<https://jinja.palletsprojects.com/en/3.0.x/templates/#list-of-builtin-filters>

Über die normalen Jinja2-Filter hinaus verfügt Ansible noch über eine Reihe eigener Filter, die in der gleichen Art und Weise in Kombination mit der Jinja2-Syntax verwendet werden können. Hierzu gehören z. B. Filter, um Variablen und Ansible Facts in bestimmte Datenstrukturen zu formatieren:

```
---
- name: a playbook to use some filter in ansible
  hosts: centos1
  tasks:
    - name: a task to get some input
      ansible.builtin.command: cat /etc/filesystems
      register: command_result

    - name: a debug task to represent
      ansible.builtin.debug:
        msg: "{{ command_result['stdout_lines'] | to_yaml }}"
```

Der Filter `to_yaml` formatiert die Ausgabe der registrierten Variable `command_result` zu YAML:

```
# ansible-playbook -i inventory jinja2_filter_ansible_2.yml

PLAY [a playbook to use some filter in ansible] **

TASK [Gathering Facts] **
ok: [centos1]

TASK [a task to get some input] **
changed: [centos1]

TASK [a debug task to represent] **
ok: [centos1] => {
```

```
"msg": "[\"xfs\", \"ext4\", \"ext3\", \"ext2\", \"nodev proc\",
        ↪ \"nodev devpts\", \"iso9660\", \"vfat\", \"hfs\",
        ↪ \"hfsplus\", \"*\"]"
}

PLAY RECAP **

centos1 : ok=3      changed=1      unreachable=0
          ↪ failed=0
```

Eine Übersicht über die Verwendung von Filtern in Ansible sowie die Ansible-spezifischen Filter finden Sie unter:

[https://docs.ansible.com/ansible/latest/user\\_guide/playbooks\\_filters.html](https://docs.ansible.com/ansible/latest/user_guide/playbooks_filters.html)

## Jinja2-Filter und Tests

- Tests überprüfen Werte auf bestimmte Eigenschaften
- zahlreiche Operatoren, z. B.  
is defined, is number, is gt, and, or, ...

Einen bestimmten Wert anhand eines Jinja2-Tests im Template einer Konfigurationsdatei festlegen:

```
{% if ansible_memtotal_mb is greater than 8192 %}
JAVA_OPTS="-Xmx4096m"
{% else %}
JAVA_OPTS="-Xmx2048m"
{% endif %}
```

Zusätzlich zu den Filtern verfügt die Jinja2-Syntax über eine Reihe von Tests, die in Kombination mit den Kontrollstrukturen wie Schleifen und If-Else-Konditionen eingesetzt werden können. Mit diesen Tests lassen sich beispielsweise Bedingungen vergleichen, überprüfen oder definieren. Wenn Werte verglichen werden sollen oder ihr Zustand überprüft werden soll, so geschieht dies mit dem Operator `is` und einer weiteren Bedingung:

```
{% if value1 is gt value2 %}
{{ value1 }} is the right one
{% else %}
value2 for the win!
{% endif %}
```

In diesem Fall werden die Werte zweier Variablen mit dem Operator `gt` (*greater than*) verglichen. Wenn der erste Wert größer ist als der zweite, trifft Bedingung eins zu. Ist der zweite Wert größer, trifft Bedingung zwei zu.

Ebenfalls kann mittels `if defined` überprüft werden, ob eine Bedingung gegeben ist, also ob z. B. eine Variable deklariert wurde oder ob sie keinen Wert hat:

```
{% if value1 is defined %}
{{ value1 }}
{% else %}
is not defined
{% endif %}
```

Die Operatoren können auch untereinander kombiniert werden, sodass Bedingungen präziser geprüft werden können.

```
{% if value1 is defined and value1 not gt 5 %}
{{ value1 }}
{% else %}
value2
{% endif %}
```

In diesem Fall trifft Bedingung eins zu, wenn `value1` definiert ist und der Wert der Variablen zudem nicht größer als fünf ist. Ansonsten trifft die zweite Bedingung zu.

### **Operatoren für Jinja2:**

Jinja2 verfügt über eine Reihe von Operatoren, welche in den Kontrollstrukturen verwendet werden können. Im Folgenden eine Auswahl:

#### **Vergleiche:**

OPERATOR:	DIE BEDINGUNG IST WAHR, WENN
<code>eq</code> ( <code>==</code> )	der Wert gleich ist.
<code>ne</code> ( <code>!=</code> )	der Wert ungleich ist.
<code>gt</code> ( <code>&gt;</code> )	der Wert größer ist.
<code>ge</code> ( <code>&gt;=</code> )	der Wert größer oder gleich ist.
<code>lt</code> ( <code>&lt;</code> )	der Wert kleiner ist.
<code>le</code> ( <code>&lt;=</code> )	der Wert kleiner oder gleich ist.

#### **Logik:**

OPERATOR:	DIE BEDINGUNG IST WAHR, WENN
<code>and</code>	eine <i>und</i> andere Bedingungen zutreffen.
<code>or</code>	eine <i>oder</i> andere Bedingungen zutreffen.
<code>not</code>	die Bedingung nicht zutrifft (kann mit <code>is</code> und <code>in</code> kombiniert werden).
<code>(expr)</code>	eine als Gruppe zusammengefasste Bedingung zutrifft ( <code>foo and bar</code> ).

#### **Weitere Operatoren:**

OPERATOR:	FUNKTION:
<code>in</code>	Ist wahr, wenn ein Wert in einem anderen enthalten ist.
<code>is</code>	Prüft auf einen Wert.
<code> </code>	Wendet einen Filter an.
<code>~</code>	Konvertiert alle Operanden in Strings und verkettet diese.

Eine Übersicht über alle verfügbaren Tests in Jinja2 finden Sie unter:

<https://jinja.palletsprojects.com/en/3.0.x/templates/#builtin-tests>



### 16.3.15 Aufgabenteil



## Aufgabenteil – Jinja2 Filter und Tests

- ① Schreiben Sie ein Playbook, in dem Sie Informationen mit Jinja2 filtern.
- ② Schreiben Sie ein Playbook mit zwei Tasks, die abhängig von bestimmten Bedingungen ausgeführt werden.

### 1. Schreiben Sie ein Playbook mit folgenden Eigenschaften:

- In einem ersten Task wird mit dem Modul `ansible.builtin.command` die Datei `/etc/filesystems` auf dem System `centos1` ausgelesen. Das Ergebnis wird in einer Variablen registriert.
- Im zweiten Task wird mit dem `ansible.builtin.debug`-Modul und dem Parameter `msg` die registrierte Variable aufgerufen. Wenden Sie bei dem Aufruf Jinja2-Filter an, sodass Sie eine kommaseparierte Liste des Inhalts der Datei `/etc/filesystems` erhalten.

### 2. Schreiben Sie ein Playbook mit zwei Tasks für den Host `suse1`, die abhängig von bestimmten Bedingungen ausgeführt werden:

- Deklarieren Sie eine Variable `value1` mit dem Wert `2` und eine Variable `value2` mit einem beliebigen Wert. Schreiben Sie einen Task, der mit dem `ansible.builtin.debug`-Modul und dem Operator `msg` den Wert der Variablen `value1` ausgibt. Die Ausführung dieses Tasks soll von einem When-Konditional abhängen, das mit der Jinja2-Syntax folgende Bedingung prüft: Der Task wird ausgeführt, wenn die Variable `value1` definiert ist und ihr Wert nicht größer als vier ist.

## 16 Ansible Playbooks und Roles

- Schreiben Sie einen zweiten Task, dessen Ausführung ebenfalls von einem When-Konditional abhängt. Dieser Task soll ausgeführt werden, wenn `value1` keinen Wert hat oder der Wert größer als vier ist. Trifft diese Bedingung zu, so soll der Task mit dem `ansible.builtin.debug`-Modul und dem Operator `msg` den Wert der Variablen `value2` ausgeben.

## Musterlösung

### 1. Schreiben Sie ein Playbook mit folgenden Eigenschaften:

- In einem ersten Task wird mit dem Modul `ansible.builtin.command` die Datei `/etc/filesystems` auf dem System `centos1` ausgelesen. Das Ergebnis wird in einer Variablen registriert.
- Im zweiten Task wird mit dem `ansible.builtin.debug`-Modul und dem Parameter `msg` die registrierte Variable aufgerufen. Wenden Sie bei dem Aufruf `Jinja2-Filter` an, sodass Sie eine kommaseparierte Liste des Inhalts der Datei `/etc/filesystems` erhalten:

Erstellen Sie das Playbook für den Host `centos1` mit den beiden Tasks, z. B. in `jinja2_filter_exercise.yml`:

Lesen Sie die Datei `/etc/filesystems` mit dem `command`-Modul aus:

```
---
- name: a playbook to filter with Jinja2
  hosts: centos1
  tasks:
    - name: the command task to read out /etc/filesystems
      ansible.builtin.command: cat /etc/filesystems
      register: command_result
    - name: the task to filter with the debugmodul
      ansible.builtin.debug:
        msg: "{{ command_result['stdout_lines'] | join(',') }}"
```

Führen Sie das Playbook nun aus, so erscheint der Inhalt von `/etc/filesystems` als kommaseparierte Liste:

```
# ansible-playbook -i /root/code/inventory
  ↪ jinja2_filter_exercise.yml

PLAY [a playbook to filter with Jinja2] **

TASK [Gathering Facts] **
ok: [centos1]

TASK [the command task to read out /etc/filesystems] **
changed: [centos1]

TASK [the task to filter with the debugmodul] **
ok: [centos1] => {
    "msg": "xfs,ext4,ext3,ext2,nodev proc,nodev
          → devpts,iso9660,vfat,hfs,hfsplus,*"
}

PLAY RECAP **
centos1 : ok=3      changed=1      unreachable=0
  ↪     failed=0
```

**2. Schreiben Sie ein Playbook mit zwei Tasks für den Host `suse1`, die abhängig von bestimmten Bedingungen ausgeführt werden:**

- Deklarieren Sie eine Variable `value1` mit dem Wert 2 und eine Variable `value2` mit einem beliebigen Wert. Schreiben Sie einen Task, der mit dem `ansible.builtin.debug`-Modul und dem Operator `msg` den Wert der Variablen `value1` ausgibt. Die Ausführung dieses Tasks soll von einem When-Konditional abhängen, das mit der Jinja2-Syntax folgende Bedingung prüft: Der Task wird ausgeführt, wenn die Variable `value1` definiert ist und ihr Wert nicht größer als vier ist.
- Schreiben Sie einen zweiten Task, dessen Ausführung ebenfalls von einem When-Konditional abhängt. Dieser Task soll ausgeführt werden, wenn `value1` keinen Wert hat oder der Wert größer als vier ist. Trifft diese Bedingung zu, so soll der Task mit dem `ansible.builtin.debug`-Modul und dem Operator `msg` den Wert der Variablen `value2` ausgeben:

Playbook mit den beiden Tasks und den jeweiligen When-Konditionalen, z. B. in `jinja2_test_exercise.yml`:

```
---
- name: a playbook to test some tests
  hosts: suse1
  vars:
    value1: '2'
    value2: 'This is value2.'
  tasks:
    - name: a task for the first condition
      ansible.builtin.debug:
        msg: "{{ value1 }}"
      when: value1 is defined and value1 is not gt '4'

    - name: a task for the second condition
      ansible.builtin.debug:
        msg: "{{ value2 }}"
      when: value1 is not defined or value1 is gt '4'
```

Ausführung des Playbooks:

```
# ansible-playbook -i /root/code/inventory
  ↪ jinja2_test_exercise.yml
```

```
PLAY [a playbook to test some tests] **

TASK [Gathering Facts] **
ok: [suse1]

TASK [a task for the first condition] **
ok: [suse1] => {
    "msg": "2"
}

TASK [a task for the second condition] **
```

```
skipping: [suse1]

PLAY RECAP **
suse1 : ok=2     changed=0    unreachable=0
      ↪ failed=0
```

Ist der Wert der Variablen `value1` geringer oder gleich vier, wird der erste Task ausgeführt und der Wert von `value1` ausgegeben. Ist der Wert von Variable `value1` nicht gesetzt oder größer als vier, wird die Variable `value2` ausgegeben.

### 16.3.16 Rollen: templates/



## Rollen: templates/

Templates in Ansible:

- dienen als Vorlage für z. B. Konfigurationsdateien
- können Variablen und Ansible Facts auslesen
- werden über das Modul template aufgerufen
- werden in der „Templating-Sprache“ Jinja2 verfasst
- Templates auf dem Controller:  
/roles/rolename/templates/example.j2

Jinja2 Template für eine vhost-Konfigurationsdatei:

```
<VirtualHost *:80>
ServerAdmin webmaster@{{ item['name'] }}
DocumentRoot {{ item['docroot'] }}
ServerName {{ item['name'] }}
ServerAlias www.{{ item['name'] }}
</VirtualHost>
```

Für den Fall, dass viele Dateien mit ähnlichem Inhalt auf den Zielsystemen gebraucht werden, kann Ansible mit Templates eingesetzt werden. Dies kann z. B. nützlich sein, wenn mehrere einander ähnliche Konfigurationsdateien für denselben Dienst auf einem oder mehreren Systemen benötigt werden. Diese Templates werden in der Templating-Sprache Jinja2 verfasst und im Verzeichnis templates/ platziert, von wo aus sie dann in einem Task mit dem Modul ansible.builtin.template aufgerufen werden können. Die Templates, die das ansible.builtin.template-Modul erzeugt, werden auf dem Controller generiert und anschließend auf die Zielsysteme kopiert.

Für die Rolle httpd bietet es sich beispielsweise an, die einzelnen Konfigurationsdateien der *Virtual Hosts* mit einem Template generieren zu lassen. Eine Vorlage, die eine einfache Konfigurationsdatei für einen virtuellen httpd-Host erstellt, könnte so aussehen:

```
# cat templates/vhost.conf.j2
<VirtualHost *:80>
ServerAdmin webmaster@{{ item['name'] }}
DocumentRoot {{ item['docroot'] }}
ServerName {{ item['name'] }}
ServerAlias www.{{ item['name'] }}
</VirtualHost>
```

Damit diese Vorlage funktioniert, müssen allerdings die in ihr verwendeten Variablen deklariert werden, sodass diese vom `ansible.builtin.template`-Modul ausgelesen werden können. Die Werte dieser Variablen werden vom Nutzer selbst deklariert und sind nicht Bestandteil der eigentlichen Rolle. Daher werden sie entweder als `host_vars` oder als `group_vars` festgelegt. Die Rolle soll auf die Gruppe `webserver` angewendet werden, weshalb die Variablen in den `group_vars` deklariert sind. Da das `template`-Modul, welches später die `vhost.conf.j2` in einem Task aufruft, auch mehrere Konfigurationsdateien in einem Task erzeugen können soll, werden die Variabldaten in einer Liste deklariert, damit der Task diese mit einem `loop` durchiterieren kann. Diese Liste befindet sich im Dictionary `httpd_vhosts`:

```
# cat /root/code/group_vars/webserver.yml

---
httpd_package_state: latest
httpd_vhosts:
  - name: b1.de
    docroot: '/var/www/b1.de/'
  - name: training.b1.de
    docroot: '/var/www/training.b1.de/'
```

Anschließend muss der Task, der das `ansible.builtin.template`-Modul beinhaltet, erstellt werden:

```
# cat tasks/main.yml

---
- include_vars: "{{ ansible_os_family }}.yml"

- name: Ensure httpd is installed in its {{ httpd_package_state }}
  ↪ state
  ansible.builtin.package:
    name: "{{ httpd_package }}"
    state: "{{ httpd_package_state }}"

- name: create the httpd vhost conf
  ansible.builtin.template:
    src: vhost.conf.j2
    dest: "{{ httpd_vhost_dir }}/{{ item['name'] }}.conf"
    loop: "{{ httpd_vhosts }}"
```

Das `ansible.builtin.template`-Modul nimmt nun die `vhost.conf.j2` und ersetzt die darin enthaltenen Variablen durch die Listenpunkte der in `group_vars/webserver.yml` deklarierten Liste, um diese in das unter der Option `dest`: angegebene Verzeichnis zu kopieren. Da der Verzeichnispfad allerdings von Distribution zu Distribution verschieden ist, muss dieser in den jeweiligen Dateien im `vars`-Verzeichnis der Rolle ergänzt werden:

Für Red Hat-Systeme:

```
---
httpd_package: httpd
httpd_vhost_dir: '/etc/httpd/conf.d/'
```

## 16 Ansible Playbooks und Roles

Für SUSE-Systeme:

```
---
```

```
httpd_package: apache2
httpd_vhost_dir: '/etc/apache2/vhosts.d/'
```

Für Debian-Systeme:

```
---
```

```
httpd_package: apache2
httpd_vhost_dir: '/etc/apache2/sites-available/'
```

Bei Aufruf der Rolle für die Gruppe `webserver` wird aus dem `vars`-Verzeichnis die entsprechende Variablendatei mit der jeweils richtigen Pfadangabe ausgelesen. Das `ansible.builtin.template`-Modul iteriert dann über die Liste in den `group_vars` und erstellt mit der Jinja2-Datei aus dem `template`-Verzeichnis eine Konfigurationsdatei, die im entsprechenden Pfad abgelegt wird.

Auf dem ersten System der Gruppe `webserver`:

```
root@ubuntu1 ~]# ls /etc/apache2/sites-available/
000-default.conf  b1.de.conf  default-ssl.conf  training.b1.de.conf
```

Mit dem Inhalt:

```
root@ubuntu1 ~]# cat /etc/apache2/sites-available/b1.de.conf
<VirtualHost *:80>
ServerAdmin webmaster@b1.de
DocumentRoot /var/www/b1.de/
ServerName b1.de
ServerAlias www.b1.de
</VirtualHost>
```

Auf dem zweiten System der Gruppe `webserver`:

```
root@centos2 ~]# ls /etc/httpd/conf.d/
autoindex.conf  b1.de.conf  README  training.b1.de.conf  userdir.conf
    ↳ welcome.conf
```

Mit dem Inhalt:

```
root@centos2 ~]# cat /etc/httpd/conf.d/training.b1.de.conf
<VirtualHost *:80>
ServerAdmin webmaster@training.b1.de

DocumentRoot /var/www/training.b1.de/
ServerName training.b1.de
ServerAlias www.training.b1.de
</VirtualHost>
```

In einem weiteren Task soll noch das in der Konfigurationsdatei beschriebene `DocumentRoot` angelegt werden. Dies erledigt das Modul `file` mit einer `loop`-Schleife, die sich ebenfalls auf die in den `group_vars` deklarierte Liste bezieht:

```
# cat tasks/main.yml

---
- include_vars: "{{ ansible_os_family }}.yml"

- name: Ensure httpd is installed in its {{ httpd_package_state }}
  & state
  ansible.builtin.package:
    name: "{{ httpd_package }}"
    state: "{{ httpd_package_state }}"

- name: create the httpd vhost conf
  ansible.builtin.template:
    src: vhost.conf.j2
    dest: "{{ httpd_vhost_dir }}/{{ item['name'] }}.conf"
    loop: "{{ httpd_vhosts }}"

- name: create a directory for docroot
  ansible.builtin.file:
    name: "{{ item['docroot'] }}"
    state: directory
    loop: "{{ httpd_vhosts }}"
```

Jetzt werden neben den Konfigurationsdateien auch die entsprechenden Verzeichnisse auf den Zielsystemen angelegt.



### 16.3.17 Aufgabenteil



## Aufgabenteil – Templates

- ① Erstellen Sie ein Template für vhost-Dateien für einen Apache2-Webserver.
- ② Erstellen Sie im Verzeichnis `group_vars` eine Liste mit den Werten für Ihr Template für die Gruppe `webserver`.
- ③ Schreiben Sie einen Task, der das erstellte Template aufruft und auf die Zielsysteme kopiert.
- ④ Schreiben Sie einen Task, der die Verzeichnisse anlegt, die in der Liste in `group_vars` deklariert wurden.
- ⑤ Deklarieren Sie die für das Ausführen des Tasks notwendigen Variablen in den Dateien in `vars/`.

1. Erstellen Sie im Verzeichnis `templates/` ein Template für vhost-Dateien mit den Einträgen `ServerName`, `DocumentRoot` und `ServerAlias` für einen Apache2-Webserver.
2. Erstellen Sie im Verzeichnis `group_vars` eine Liste mit den Werten für die Einträge in Ihrem Template für die Gruppe `webserver`.
3. Schreiben Sie einen Task, der das erstellte Template aufruft, mit einer `loop`-Schleife durch Ihre Liste in `group_vars` iteriert und die Konfigurationsdatei auf die Zielsysteme kopiert.
4. Schreiben Sie einen Task, der die Verzeichnisse anlegt, die in der Liste in `group_vars` deklariert wurden.
5. Deklarieren Sie die für das Ausführen des Tasks notwendigen Variablen in Abhängigkeit der jeweiligen Distribution in den Dateien in `vars/`.

## Musterlösung

1. Erstellen Sie im Verzeichnis `templates/` ein Template für `vhost`-Dateien mit den Einträgen `ServerName`, `DocumentRoot` und `ServerAlias` für einen Apache2-Webserver:

Legen Sie die Jinja2-Datei im `templates`-Verzeichnis an, z. B. `vhost.conf.j2`, und fügen Sie den Text und die Variablen für die Listenzugriffe ein:

```
<VirtualHost *:80>
ServerName {{ item['name'] }}
DocumentRoot {{ item['docroot'] }}
ServerAlias www.{{ item['name'] }}
</VirtualHost>
```

2. Erstellen Sie im Verzeichnis `group_vars` eine Liste mit den Werten für die Einträge in Ihrem Template für die Gruppe `webserver`:

Da Sie die Liste für eine Gruppe anlegen möchten, müssen Sie dies in der Datei für die Gruppe `webserver` im Verzeichnis `group_vars/` tun. Fügen Sie die Liste also der Datei `group_vars/webserver.yml` hinzu. Der Name des Dictionary, das die Liste beinhaltet, ist der Name, der später als Variable referenziert werden kann:

```
apache2_package_state: latest
apache_vhosts:
  - name: mysite.de
    docroot: '/var/www/mysite.de/'
  - name: home.b1.de
    docroot: '/var/www/home.b1.de/'
```

3. Schreiben Sie einen Task, der das erstellte Template aufruft, mit einer `loop`-Schleife durch Ihre Liste in `group_vars` iteriert und die Konfigurationsdatei auf die Zielsysteme kopiert:

Fügen Sie den Task mit dem `ansible.builtin.template`-Modul und der `loop`-Schleife der Datei `tasks/main.yml` hinzu:

```
---
- include_vars: "{{ ansible_os_family }}.yml"
- name: ensure that the "{{ apache2_package_state }}" version of
  ↪ the apache2 package is there
  ansible.builtin.package:
    name: "{{ apache2_package }}"
    state: "{{ apache2_package_state }}"
  notify:
    - Start apache2

- name: create the apache vhost conf
  ansible.builtin.template:
    src: vhost.conf.j2
    dest: "{{ apache_vhost_dir }}/{{ item['name'] }}.conf"
  loop: "{{ apache_vhosts }}"
```

**4. Schreiben Sie einen Task, der die Verzeichnisse anlegt, die in der Liste in `group_vars` deklariert wurden:**

Fügen Sie den Task mit dem Modul `file`, der Option `state: directory` und einer `loop`-Schleife der Datei `tasks/main.yml` hinzu:

```
---
- include_vars: "{{ ansible_os_family }}.yml"
- name: ensure that the "{{ apache2_package_state }}" version of
      → the apache2 package is there
  ansible.builtin.package:
    name: "{{ apache2_package }}"
    state: "{{ apache2_package_state }}"
  notify:
    - Start webserver

- name: create the apache vhost conf
  ansible.builtin.template:
    src: vhost.conf.j2
    dest: "{{ apache_vhost_dir }}/{{ item['name'] }}.conf"
  loop: "{{ apache_vhosts }}"

- name: create a directory for docroot
  ansible.builtin.file:
    name: "{{ item['docroot'] }}"
    state: directory
  loop: "{{ apache_vhosts }}"
```

**5. Deklarieren Sie die für das Ausführen des Tasks notwendigen Variablen in Abhängigkeit der jeweiligen Distribution in den Dateien in `vars`:**

Da sich die Pfade der Konfigurationsdateien von `apache2` bzw. `httpd` in den jeweiligen Distributionen unterscheiden, müssen diese als Variablen an die Rolle übergeben werden. Es muss also in den distributionsspezifischen Dateien in `vars/` die Variable `apache_vhost_dir` mit dem jeweiligen Pfad als Wert deklariert werden. Fügen Sie die Variablen den jeweiligen Dateien hinzu:

Für Red Hat-Systeme `vars/RedHat.yml`:

```
---
apache2_package: httpd
apache_vhost_dir: '/etc/httpd/conf.d/'
```

Für SUSE-Systeme `vars/Suse.yml`:

```
---
apache2_package: apache2
apache_vhost_dir: '/etc/apache2/vhosts.d/'
```

Für Debian-Systeme `vars/Debian.yml`:

```
---
apache2_package: apache2
apache_vhost_dir: '/etc/apache2/sites-available/'
```

## 16 Ansible Playbooks und Roles

Diese Variablen werden nun über die `include_vars`-Anweisung in der `tasks/main.yml` distributionsspezifisch ausgewertet, wodurch vom `template`-Modul der jeweils korrekte Pfad aufgerufen wird.

### 16.3.18 Rollen: templates/ und tasks/



## Rollen: templates/ und tasks/

- leere Listen in der defaults/main.yml deklarieren
- Tasks in eigene YAML-Dateien auslagern
  - mit include\_tasks wieder einbinden

defaults/main.yml

```
httpd_package_state: present
apache_vhosts: []
```

### Ausgelagerte Tasks in tasks/

main.yml  
vhost.yml

### Tasks in tasks/main.yml einfügen:

```
- include_tasks: vhost.yml
loop: "{{ apache_vhosts }}"
```

Wenn die bisher beschriebene Rolle ausgeführt wird, installiert sie den Webserver apache2 auf den im Inventory eingetragenen Systemen – sofern im Playbook angegeben – und kopiert mit dem ansible.builtin.template-Modul Konfigurationsdateien in die entsprechenden Pfade der Systeme der Gruppe webserver. Wird die Rolle jedoch für ein System angewendet, das nicht zur Gruppe webserver gehört, scheitert der Task, der die Konfigurationsdateien kopiert, da die loop-Schleife die Variable apache\_vhosts mit der Liste nicht auflösen kann:

```
TASK [httpd : create the apache vhost conf] **
fatal: [suse2]: FAILED! => {"msg": "'apache_vhosts' is undefined"}
```

An diesem Punkt wird die Ausführung des die Rolle enthaltenden Playbooks mit einem Fehler abgebrochen. Um dies zu verhindern, kann in der defaults/main.yml dieselbe Variable mit einer leeren Liste gesetzt werden:

```
# cat defaults/main.yml
---
httpd_package_state: present
apache_vhosts: []
```

Ist diese leere Liste deklariert, so liest die loop-Schleife diese aus und der Task wird nicht ausgeführt, wodurch das Ausführen des Playbooks nicht unterbrochen wird. Wird

## 16 Ansible Playbooks und Roles

jedoch eine Liste in den `host-` oder `group_vars` deklariert und werden diese Hosts oder Gruppen im Playbook angesprochen, so überschreiben diese die Variable in der `defaults/main.yml` und der Task wird für diese Systeme ausgeführt.

### `include_tasks`

Um zu verhindern, dass die Liste mit Tasks in der `tasks/main.yml` immer länger und unübersichtlicher wird, können diese in eigene YAML-Dateien im `tasks`-Verzeichnis ausgelagert werden:

```
# cat tasks/vhost.yml

---
- name: create the apache vhost conf
  ansible.builtin.template:
    src: vhost.conf.j2
    dest: "{{ apache_vhost_dir }}/{{ item['name'] }}.conf"

- name: create a directory for docroot
  ansible.builtin.file:
    name: "{{ item['docroot'] }}"
    state: directory
```

Anschließend können die beiden Tasks in der `task/main.yml` mit dem Modul `include_tasks`: unter Angabe der Datei `tasks/vhost.yml` ersetzt werden:

```
# cat tasks/main.yml

---
- include_vars: "{{ ansible_os_family }}.yml"

- name: Ensure httpd is installed in its {{ httpd_package_state }}
  ↪ state
  ansible.builtin.package:
    name: "{{ httpd_package }}"
    state: "{{ httpd_package_state }}"

- include_tasks: vhost.yml
  loop: "{{ apache_vhosts }}"
```

Da beide Tasks in der `tasks/vhost.yml` mit einer Schleife durch die in `group_vars` deklarierte Liste iterieren sollen, kann die `loop`-Schleife dem `include_vars`: angehängt werden. So muss dies nicht an jedem Task einzeln geschehen.

### 16.3.19 Aufgabenteil



## Aufgabenteil – templates/ und tasks/

- ① Deklarieren Sie eine leere Liste in der defaults/main.yml.
- ② Überführen Sie die Tasks der Rolle apache2\_role, die die Vhost-Konfiguration betreffen, in eine eigene Task-Datei.

1. Deklarieren Sie die Liste apache\_vhosts aus group\_vars/webserver.yml als leere Liste in der defaults/main.yml der Rolle apache2\_role.
2. Überführen Sie die Tasks der Rolle apache2\_role, die die Vhost-Konfiguration betreffen, in eine eigene Task-Datei. Binden Sie diese mit dem Modul include\_vars in die tasks/main.yml ein.

## Musterlösung

1. Deklarieren Sie die Liste `apache_vhosts` aus `group_vars/webserver.yml` als leere Liste in der `defaults/main.yml` der Rolle `apache2_role`:

Fügen Sie die leere Liste `apache_vhosts` der Datei `defaults/main.yml` hinzu:

```
---
```

```
apache2_package_state: present
apache_vhosts: []
```

2. Überführen Sie die Tasks der Rolle `apache2_role`, die die Vhost-Konfiguration betreffen, in eine eigene Task-Datei. Binden Sie diese mit dem Modul `include_vars` in die `tasks/main.yml` ein:

Erstellen Sie zunächst die Datei `tasks/vhost.yml`. Kopieren Sie die beiden Tasks aus der `tasks/main.yml` hinüber, wobei Sie die `loop`-Schleifen weglassen:

```
---
```

```
- name: create the apache vhost conf
  ansible.builtin.template:
    src: vhost.conf.j2
    dest: "{{ apache_vhost_dir }}/{{ item['name'] }}.conf"

- name: create a directory for docroot
  ansible.builtin.file:
    name: "{{ item['docroot'] }}"
    state: directory
```

Anschließend binden Sie die Datei `tasks/vhost.yml` mit dem Modul `include_tasks` in die `tasks/main.yml` ein und fügen die `loop`-Schleife an:

```
---
```

```
- include_vars: "{{ ansible_os_family }}.yml"

- name: ensure that the "{{ apache2_package_state }}" version of
  ↪ the apache2 package is there
  ansible.builtin.package:
    name: "{{ apache2_package }}"
    state: "{{ apache2_package_state }}"

- include_tasks: vhost.yml
  loop: "{{ apache_vhosts }}"
```

## 16.4 Aufruf von Rollen



### Aufruf von Rollen

#### Aufruf von Rollen im Playbook

```
---
- hosts: all
  roles:
    - baseline
    - httpd
    - website
```

#### Das httpd-Playbook

```
---
- name: a playbook to include the role httpd
  hosts: webserver
  roles:
    - httpd
```

Nachdem nun die Rolle eingerichtet ist, muss sie nur noch ausgeführt werden. Um eine Rolle auszuführen, wird diese in einem Playbook mit der Option `roles:` in Listenform aufgerufen:

```
# cat role_httpd_playbook.yml
---
- name: a playbook to include the role httpd
  hosts: webserver
  roles:
    - httpd
```

Die hier eingefügten Rollen werden dann der Reihe nach auf die unter `hosts:` angegebenen Zielsysteme angewendet. Anschließend wird das Playbook ganz normal mit dem Befehl `ansible-playbook` ausgeführt:

```
# ansible-playbook -i inventory role_httpd_playbook.yml
```

## 16 Ansible Playbooks und Roles

Der Befehl lädt die in der Rolle deklarierten Variablen und die erstellten Tasks und führt diese für die Zielsysteme aus:

```
PLAY [a playbook to include the role httpd] **

TASK [Gathering Facts] **
ok: [ubuntu1]
ok: [centos2]

TASK [httpd : include_vars] **
ok: [centos2]
ok: [ubuntu1]

TASK [httpd : Ensure httpd is installed in its latest state] **
ok: [ubuntu1]
ok: [centos2]

TASK [httpd : include_tasks] **
included: /root/code/roles/httpd/tasks/vhost.yml for centos2, ubuntu1
    ↪ => (item={'docroot': u'/var/www/b1.de/', 'name': u'b1.de'})
included: /root/code/roles/httpd/tasks/vhost.yml for centos2, ubuntu1
    ↪ => (item={'docroot': u'/var/www/training.b1.de/', 'name':
    ↪ u'training.b1.de'})

TASK [httpd : create the apache vhost conf] **
ok: [ubuntu1]
ok: [centos2]

TASK [httpd : create a directory for docroot] **
ok: [ubuntu1]
ok: [centos2]

TASK [httpd : create the apache vhost conf] **
ok: [ubuntu1]
ok: [centos2]

TASK [httpd : create a directory for docroot] **
ok: [ubuntu1]
ok: [centos2]

PLAY RECAP **
centos2                  : ok=9      changed=0      unreachable=0
    ↪ failed=0
ubuntu1                  : ok=9      changed=0      unreachable=0
    ↪ failed=0
```

Der in diesem Playbook ausgeführten Rolle können noch beliebig viele hinzugefügt werden. Über die Variablen in der defaults/main.yml sowie host- und group\_vars können zudem Filter eingebaut werden, sodass bestimmte Rollen oder bestimmte Teile von Rollen ausschließlich auf bestimmten Systemen ausgeführt werden.

#### 16.4.1 Aufgabenteil



## Aufgabenteil – Ausführen von Rollen

- ① Schreiben Sie ein Playbook, das die Rolle apache2\_role für die Gruppe webserver ausführt.
- ② Führen Sie das neue Playbook aus.

1. Schreiben Sie ein Playbook, das die Rolle apache2\_role für die Gruppe webserver ausführt.
2. Führen Sie das neue Playbook aus.

## Musterlösung

**1. Schreiben Sie ein Playbook, das die Rolle apache2 für die Gruppe webserver ausführt:**

Erstellen Sie die Datei `role_apache2_playbook.yml`. Tragen Sie bei `hosts: webserver` ein. Fügen Sie den entsprechenden Befehl `roles:` und den Namen der Rolle ein, damit die Rolle geladen wird:

```
---
- name: a playbook to include the role apache2_role
  hosts: webserver
  roles:
    - apache2_role
```

**2. Führen Sie das neue Playbook aus:**

Führen Sie den Befehl `ansible-playbook` unter Angabe des Inventorys und des Playbooks aus, das die Rolle enthält:

```
# ansible-playbook -i /root/code/inventory
  ↪ role_apache2_playbook.yml

PLAY [a playbook to include the role apache2_role]
  ↪ ****
TASK [Gathering Facts]
  ↪ ****
ok: [ubuntu1]
ok: [centos2]

TASK [apache2_role : include_vars]
  ↪ ****
ok: [centos2]
ok: [ubuntu1]

TASK [apache2_role : ensure that the "present" version of the
      ↪ apache2 package is there] ****
changed: [ubuntu1]
changed: [centos2]

TASK [apache2_role : ensure that the service is up and running]
  ↪ ****
changed: [centos2]
changed: [ubuntu1]

TASK [apache2_role : include_tasks]
  ↪ ****
included: /root/code/roles/apache2/tasks/vhost.yml for centos2 =>
  ↪ (item={u'docroot': u'/var/www/b1.de/', u'name': u'b1.de'})
included: /root/code/roles/apache2/tasks/vhost.yml for centos2 =>
  ↪ (item={u'docroot': u'/var/www/training.b1.de/', u'name':
  ↪ u'training.b1.de'})
included: /root/code/roles/apache2/tasks/vhost.yml for ubuntu1 =>
  ↪ (item={u'docroot': u'/var/www/b1.de/', u'name': u'b1.de'})
```

```

included: /root/code/roles/apache2/tasks/vhost.yml for ubuntul =>
  ↪ (item={u'docroot': u'/var/www/training.bl.de/', u'name':
  ↪ u'training.bl.de'})

TASK [apache2_role : create the apache vhost conf]
  ↪ ****
ok: [centos2]
ok: [ubuntul]

TASK [apache2_role : create a directory for docroot]
  ↪ ****
ok: [centos2]
ok: [ubuntul]

TASK [apache2_role : create the apache vhost conf]
  ↪ ****
ok: [centos2]
ok: [ubuntul]

TASK [apache2_role : create a directory for docroot]
  ↪ ****
ok: [centos2]
ok: [ubuntul]

RUNNING HANDLER [apache2_role : restart webserver]
  ↪ ****
changed: [centos2]
changed: [ubuntul]
PLAY RECAP ****
centos2 : ok=11    changed=3      unreachable=0
  ↪     failed=0
ubuntul : ok=11    changed=3      unreachable=0
  ↪     failed=0

```

Das Playbook lädt die Rolle und führt die darin enthaltenen Tasks aus. Dabei werden die Zielsysteme, die nicht der Gruppe `webserver` angehören, in dem Task, der das Webserverpaket installiert, übersprungen, da die `when`-Kondition nicht zutrifft. In den weiteren Tasks, die über das `include_vars` in die `tasks/main.yml` integriert sind, sorgt die Iteration der `loop`-Schleife dafür, dass die Tasks nur für die Zielsysteme ausgeführt werden, für die eine Liste zum Iterieren deklariert wurde. Dies sind in diesem Fall die Systeme der Gruppe `webserver`, deren Liste in den `group_vars` deklariert wurde. Für alle anderen Systeme gilt die in der `defaults/main.yml` deklarierte leere Liste, die dafür sorgt, dass der Task nicht ausgeführt wird.



## Aufruf mit Übergabe von Parametern

### Aufruf mit Parametern

```
---
- hosts: all
  roles:
    - role: baseline
      users:
        - name: 'paul'
          groups: [ 'wheel' ]
        - name: 'peter'
        - name: 'mary'
          groups: [ 'wheel', 'video' ]
```

Mit der Übergabe von Parametern an Rollen in Ansible ist es möglich, Rollen mit eigenen Daten auszuführen. So wird eine Rolle vielseitig und systemübergreifend nutzbar. So kann z. B. eine Rolle, die Benutzer anlegt, dadurch in verschiedenen Landschaften mit verschiedenen Anforderungen eingesetzt werden. Die Liste mit den anzulegenden Benutzern ist in diesem Fall nicht Teil der Rolle, sondern wird beim Ausführen eingelesen, ihr Inhalt wird der Rolle als Parameter übergeben. Solche Parameter werden direkt in dem Playbook, das die Rolle einbindet, als Variablen der Rolle deklariert:

```
---
- hosts: centos2
  roles:
    - role: baseline
      users:
        - name: 'paul'
          groups: [ 'wheel' ]
        - name: 'peter'
        - name: 'mary'
          groups: [ 'wheel', 'video' ]
```

In der Struktur der Rolle `baseline` wird der Task erstellt, der auf die Variable `users` zugreifen soll:

```
baseline/
  tasks/
    main.yml
```

Im Folgenden wird der Task in der `main.yml` eingetragen:

```
- name: create users
  user:
    name: "{{ item['name'] }}"
    groups: "{{ item['groups'] | default('omit') }}"
  loop: "{{ users }}"
```

Der Task iteriert mit `loop` durch die Liste `users`, die als Variable im Playbook direkt deklariert wird. Dabei wird für den Parameter `groups` der Jinja2-Filter `default('omit')` angewendet, der dafür sorgt, dass der Parameter ausgelassen wird, wenn kein Wert für die Variable deklariert wurde. Im Beispiel trifft dies für `peter` zu:

```
# ansible-playbook -i /root/code/inventory code/parameter.yml

PLAY [playbook to create users with a role] **

TASK [Gathering Facts] **
ok: [centos2]

TASK [b1-training : create users] **
changed: [centos2] => (item={'name': u'paul', 'groups': [u'wheel']})
changed: [centos2] => (item={'name': u'peter'})
changed: [centos2] => (item={'name': u'mary', 'groups': [u'wheel',
  ↪ u'video']})

PLAY RECAP **
centos2                  : ok=2      changed=1      unreachable=0
                           ↪ failed=0
```

Durch das Deklarieren der einzusetzenden Werte als Variablen im Playbook kann die Rolle vielseitig benutzt werden. In diesem Beispiel dient die Rolle dazu, Benutzer anzulegen und ihnen Gruppen zuzuweisen. Die Benutzer und Gruppen werden dabei jedoch individuell im Playbook festgelegt und sind somit nicht Teil der Rolle. Ein solches Vorgehen empfiehlt sich immer dann, wenn die Werte, die in der Rolle verwendet werden sollen, nicht immer dieselben sind.

## Überschreiben von Variablen

Da die als Parameter an die Rolle übergebenen Variablen alle anderen Variablen mit dem selben Namen überschreiben, bietet sich hier die Möglichkeit an, die Werte, die beispielsweise im `/vars`-Verzeichnis der Rolle deklariert wurden, zu überschreiben und die Rolle mit anderen Werten auszuführen.

## 16 Ansible Playbooks und Roles

Wurde die Variable `httpd_package` in den Host-, oder Groupvars, dem Inventory oder der Rolle definiert, aber die Rolle soll mit einem speziellen Webserver-Paket ausgeführt werden, so kann dieser Wert als Parameter an die Rolle übergeben werden. Dies überschreibt alle anderen Variablen `httpd_package`:

```
---
- hosts: all
  roles:
    - role: httpd
      httpd_package: b1-apache2-custom
```

Nun wird die Rolle `httpd` den Wert `b1-apache2-custom` für die Variable `httpd_package` deklarieren, und alle anderen Deklarationen überschreiben, die in der Rolle gemacht wurden.

### 16.5.1 Aufgabenteil



## Aufgabenteil – Übergabe von Parametern

Überschreiben Sie die Variable `apache2_package_state` für die Rolle `apache2_role` mit dem Wert `latest`.

Überschreiben Sie die Variable `apache2_package_state` für die Rolle `apache2_role` mit dem Wert `latest`.

## Musterlösung

**Überschreiben Sie die Variable `apache2_package_state` für die Rolle `apache2_role` mit dem Wert `latest`:**

Fügen Sie der Rolle im entsprechenden Playbook (z. B. `/root/code/role_apache2_playbook.yml`) die Variable `apache2_package_state` mit dem neuen Wert hinzu:

```
---
- name: a playbook to include the role apache2
  hosts: all
  roles:
    - role: apache2_role
      apache2_package_state: latest
```

Nun wird der Wert `present`, der in der `default/main.yml` deklariert ist, von der Variablen `apache2_package_state` mit dem Wert `latest` überschrieben.

Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de

## 16.6 Ansible Galaxy – Rollen



# Ansible Galaxy – Rollen

Erstellen der Ordnerstruktur mit ansible-galaxy

```
# ansible-galaxy init ansible-schulung
```

```
roles/ansible-schulung/
|-- README.md
|-- defaults
|   `-- main.yml
|-- files
|-- handlers
|   `-- main.yml
|-- meta
|   `-- main.yml
|-- tasks
|   `-- main.yml
[...]
```

B1 Systems GmbH      Terraform & Ansible Grundlagen      229 / 279

Bei Ansible Galaxy handelt es sich um eine Plattform für den Austausch von Ansible Rollen. Zum einen wird die Möglichkeit geboten, fertige Rollen herunterzuladen und selber einzusetzen, zum anderen können die eigenen Rollen über Ansible Galaxy verbreitet werden. Ansible Galaxy bietet mit dem gleichnamigen Werkzeug `ansible-galaxy` ein Programm zum Bearbeiten und Organisieren von Rollen.

### Rollen verwalten mit Ansible Galaxy

Mit dem Befehl `ansible-galaxy` können Sie sich automatisiert die Verzeichnisstruktur einer Rolle samt Unterverzeichnissen erstellen lassen. Rufen Sie hierfür den Befehl `ansible-galaxy` mit dem Parameter `init` und dem gewünschten Namen der Rolle auf:

```
# ansible-galaxy init ansible-schulung
```

## 16 Ansible Playbooks und Roles

Das Programm `ansible-galaxy` legt daraufhin die Verzeichnisse, eine `README.md` und Vorlagen für die jeweiligen `main.yml`-Dateien an:

```
# tree roles/ansible-schulung/
roles/ansible-schulung/
|-- README.md
|-- defaults
|   '-- main.yml
|-- files
|-- handlers
|   '-- main.yml
|-- meta
|   '-- main.yml
|-- tasks
|   '-- main.yml
|-- templates
|-- tests
|   |-- inventory
|   '-- test.yml
`-- vars
    '-- main.yml
```

8 directories, 8 files

Die YAML-Dateien sind automatisch mit einem Kommentar versehen, der ihre Funktion erläutert:

```
# for i in $(find ansible-schulung -name "*.yml"); do echo "$i \n"
  ↪ $(awk '{print $0}' $i)\n"; done

ansible-schulung/tasks/main.yml
---
# tasks file for ansible-schulung

ansible-schulung/vars/main.yml
---
# vars file for ansible-schulung

ansible-schulung/handlers/main.yml
---
# handlers file for ansible-schulung

ansible-schulung/meta/main.yml
galaxy_info:
  author: your name
  description: your description
  company: your company (optional)

  # If the issue tracker for your role is not on github, uncomment the
  # next line and provide a value
  # issue_tracker_url: http://example.com/issue/tracker

  # Some suggested licenses:
  # - BSD (default)
  # - MIT
```

```

# - GPLv2
# - GPLv3
# - Apache
# - CC-BY
license: license (GPLv2, CC-BY, etc)

min_ansible_version: 2.4

# If this a Container Enabled role, provide the minimum Ansible
#   ↪ Container version.
# min_ansible_container_version:

# Optionally specify the branch Galaxy will use when accessing the
#   ↪ GitHub
# repo for this role. During role install, if no tags are available,
# Galaxy will use this branch. During import Galaxy will access
#   ↪ files on
# this branch. If Travis integration is configured, only
#   ↪ notifications for this
# branch will be accepted. Otherwise, in all cases, the repo's
#   ↪ default branch
# (usually master) will be used.
#github_branch:

#
# Provide a list of supported platforms, and for each platform a
#   ↪ list of versions.
# If you don't wish to enumerate all versions for a particular
#   ↪ platform, use 'all'.
# To view available platforms and versions (or releases), visit:
# https://galaxy.ansible.com/api/v1/platforms/
#
# platforms:
# - name: Fedora
#   versions:
#     - all
#     - 25
# - name: SomePlatform
#   versions:
#     - all
#     - 1.0
#     - 7
#     - 99.99

galaxy_tags: []
# List tags for your role here, one per line. A tag is a keyword
#   ↪ that describes
# and categorizes the role. Users find roles by searching for
#   ↪ tags. Be sure to
# remove the '[]' above, if you add tags to this list.
#
# NOTE: A tag is limited to a single word comprised of
#   ↪ alphanumeric characters.
#       Maximum 20 tags per role.

```

## 16 Ansible Playbooks und Roles

```
dependencies: []
  # List your role dependencies here, one per line. Be sure to remove
  # the '[]' above,
  # if you add dependencies to this list.

ansible-schulung/tests/test.yml
---
- hosts: localhost
  remote_user: root
  roles:
    - ansible-schulung

ansible-schulung/defaults/main.yml
---
# defaults file for ansible-schulung
```

Mit dem Parameter `search` können Sie Ansible Galaxy nach Rollen durchsuchen:

```
# ansible-galaxy search apache2
```

Found 235 roles matching your search:

Name	Description
infinitum.php	PHP installation role.
aalaesar.install_nextcloud	Add a new Nextcloud instance in your infrastructure. The role manages dependencies and initial configuration.
adamstraube.Ansible-Container-Webserver-Demo	Apache2 and PHP Container Demo
ajgarlag.apache2	Ansible role to install and configure apache2
ajitesh17.apache	Installing apache2
[...]	

Mit dem Parameter `install` können Sie die gefundenen Rollen auf Ihr System herunterladen und installieren:

```
# ansible-galaxy install ajitesh17.apache -p /root/code/roles
- downloading role 'apache', owned by ajitesh17
- downloading role from
  ↗ https://github.com/ajitesh17/apache/archive/master.tar.gz
- extracting ajitesh17.apache to /root/code/roles/ajitesh17.apache
- ajitesh17.apache (master) was installed successfully
```

Mit der Option `-p` (`--roles-path=`) lässt sich ein spezifischer Zielpfad für die Parameter von `ansible-galaxy` festlegen. Die Rolle wurde heruntergeladen und nach `/root/code/roles/` installiert:

```
# tree /root/code/roles/ajitesh17.apache
/root/code/roles/ajitesh17.apache
|-- README.md
|-- defaults
|   '-- main.yml
```

```

|-- handlers
|   '-- main.yml
|-- meta
|   '-- main.yml
|-- tasks
|   '-- main.yml
|-- templates
|   '-- virtualhost.conf
|-- tests
|   |-- inventory
|   '-- test.yml
`-- vars
    '-- main.yml

7 directories, 9 files

```

Mit dem Parameter `list` werden alle Rollen in einem Verzeichnis aufgelistet:

```
# ansible-galaxy list -p /root/code/roles
- ansible-schulung, (unknown version)
- ajitesh17.apache, master
- apache2, (unknown version)
```

Zum Entfernen heruntergeladener Rollen verwenden Sie den Parameter `remove`:

```
# ansible-galaxy remove ajitesh17.apache -p /root/code/roles
- successfully removed ajitesh17.apache
```

### Die Parameter von `ansible-galaxy`

PARAMETER:	FUNKTION:
--help	zeigt eine Erläuterung und listet die Parameter und Optionen auf
search	durchsucht Ansible Galaxy nach Rollen
info	gibt Informationen über die angegebene Rolle aus
install	lädt die angegebene Rolle herunter
list	gibt eine Liste der von Ansible Galaxy installierten Rollen aus
init	erzeugt eine neue Rollenstruktur auf dem eigenen System
remove	entfernt heruntergeladene Rollen wieder
-p, --roles-path=	spezifischen Pfad angeben

### Die folgenden Parameter setzen einen GitHub-Account voraus

PARAMETER:	FUNKTION:
login	Anmeldung am GitHub, um die folgenden Befehle mit <code>ansible-galaxy</code> nutzen zu können
setup	lässt Einstellungen für Galaxy und Quelle festlegen
import	importiert Rollen in die Galaxy
delete	entfernt Rollen aus der Galaxy

## 16 Ansible Playbooks und Roles

Einen Überblick, weiterführende Dokumentation und nach Themen geordnete Rollen finden Sie auf:

<https://galaxy.ansible.com/>

### 16.6.1 Aufgabenteil



## Aufgabenteil – Ansible-Galaxy

- ① Erstellen Sie eine eigene Rolle mit Ansible-Galaxy.
- ② Suchen Sie eine Rolle in der Ansible-Galaxy und installieren Sie diese.
- ③ Lassen Sie sich die installierte Rolle anzeigen und entfernen Sie sie anschließend wieder.

1. Erstellen Sie eine eigene Rolle mit Ansible-Galaxy.
2. Suchen Sie eine Rolle in der Ansible-Galaxy und installieren Sie diese im Verzeichnis `/root/code/roles/`.
3. Lassen Sie sich die installierte Rolle anzeigen und entfernen Sie sie anschließend wieder.

## Musterlösung

### 1. Erstellen Sie eine eigene Rolle mit Ansible-Galaxy:

Führen Sie ansible-galaxy mit dem Parameter `init` unter Angabe des Zielverzeichnisses aus:

```
# ansible-galaxy init /root/code/roles/b1-training
- /root/code/roles/b1-training was created successfully
```

ansible-galaxy hat Ihnen nun die folgende (leere) Rolle angelegt:

```
# tree /root/code/roles/b1-training
/root/code/roles/b1-training
|-- README.md
|-- defaults
|   '-- main.yml
|-- files
|-- handlers
|   '-- main.yml
|-- meta
|   '-- main.yml
|-- tasks
|   '-- main.yml
|-- templates
|-- tests
|   |-- inventory
|   '-- test.yml
`-- vars
    '-- main.yml
```

8 directories, 8 files

### 2. Suchen Sie eine Rolle in der Ansible-Galaxy und installieren Sie diese im Verzeichnis `/root/code/roles/`:

Suchen Sie mit dem Befehl ansible-galaxy und dem Parameter `search` nach einer Rolle, die den Webserver nginx installieren soll:

```
# ansible-galaxy search nginx
```

Found 1224 roles matching your search. Showing first 1000.

Name	Description
---	-----
[...]	

Wählen Sie eine der gefundenen Rollen und installieren Sie diese mit dem Befehl ansible-galaxy, dem Parameter `install` und der Option `-p` in das Verzeichnis `/root/code/roles/`:

```
# ansible-galaxy install iaroslavb.nginx -p /root/code/roles/
- downloading role 'nginx', owned by iaroslavb
- downloading role from https://github.com/iaroslavb/
  ↗ ansible-role-nginx/archive/master.tar.gz
- extracting iaroslavb.nginx to /root/code/roles/iaroslavb.nginx
- iaroslavb.nginx (master) was installed successfully
```

ansible-galaxy hat nun die Rolle iaroslavb.nginx installiert:

```
# tree /root/code/roles/iaroslavb.nginx
/root/code/roles/iaroslavb.nginx
|-- LICENSE
|-- README.md
|-- defaults
|   '-- main.yml
|-- handlers
|   '-- main.yml
|-- meta
|   '-- main.yml
|-- tasks
|   '-- main.yml
|-- tests
|   |-- inventory
|   |-- test.yml
|   '-- vagrant
|       |-- Vagrantfile
|       |-- ansible.cfg
|       |-- config.yml
|       |-- deploy-keys.yml
|       |-- inventory
|       '-- test.yml
`-- vars
    '-- main.yml
```

7 directories, 15 files

### 3. Lassen Sie sich die installierte Rolle anzeigen und entfernen Sie sie anschließend wieder:

Führen Sie den Befehl `ansible-galaxy` mit dem Parameter `list` und dem Pfad Ihres Rollenverzeichnisses unter der Option `-p` aus:

```
# ansible-galaxy list -p /root/code/roles
- b1-training, (unknown version)
- iaroslavb.nginx, master
- apache2, (unknown version)
```

ansible-galaxy listet Ihnen daraufhin alle Rollen aus diesem Verzeichnis auf.

Um die Rolle wieder zu entfernen, führen Sie `ansible-galaxy` mit dem Parameter `remove` unter Angabe des Rollennamens und des Pfades mit der Option `-p` aus:

```
# ansible-galaxy remove iaroslavb.nginx -p /root/code/roles
- successfully removed iaroslavb.nginx
```

## 16.7 Ansible Galaxy – Collections



# Ansible Galaxy – Collections

- Umfangreiche Verzeichnisstruktur von Collections

Erstellen einer Collection mit ansible-galaxy

```
# ansible-galaxy collection init <namespace.collection>
```

Verzeichnisstruktur von b1.schulung:

```
/root/.ansible/collections/ansible_collections/b1/
|-- schulung
    |-- docs
    |-- galaxy.yml
    |-- plugins
    |-- |-- README.md
    |-- README.md
    |-- roles
```

Collections können mehr an Funktionalität umfassen als nur die Bereitstellung von Modulen. Collections verfügen wie auch Rollen über eine umfassende Verzeichnisstruktur, in der der gesamte funktionale Code – dies meint alles außer host\_vars, group\_vars, Inventory usw. – einer Ansible-Umgebung hinterlegt werden kann. Im Prinzip handelt es sich dabei um die Weiterentwicklung der Rollenstruktur mit einer Erweiterung auf Plugins, Module und Rollen selbst. Wie auch Rollen werden Collections über die Plattform Ansible Galaxy verwaltet und distribuiert. Sie können Collections von dort herunterladen oder auch Ihre eigenen Collections erstellen und in der Galaxy verfügbar machen. Collections haben die Aufgabe, eine Struktur zur Verfügung zu stellen, die entweder schon alle benötigten Komponenten mitliefert oder selbstständig dafür sorgt, dass fehlende Abhängigkeiten automatisch nachgeladen werden. Die dafür benötigten Komponenten sind in einer Verzeichnisstruktur organisiert.



# Collections – Verzeichnisstruktur

- README.md
- docs
  - enthält allgemeine Infos über die Collection
- galaxy.yml
  - die einzige zwingend erforderliche Datei
  - enthält wichtige Infos über Version, Autor, usw.
- plugins
- roles
- test
- meta/runtime.yml
  - enthält Informationen über die Runtime, z. B. spezielle Version von ansible-core

Ansible Collections verfügen über eine umfangreiche Verzeichnisstruktur, in der alle notwendigen Komponenten und Informationen mitgeliefert werden können. Diese Verzeichnisstruktur ist allerdings bis auf die `galaxy.yml` nicht verpflichtend, sondern kommt je nach Bedarf zur Anwendung.

Der Standardpfad für Collections ist Folgender:

```
/home/user/.ansible/collections/ansible_collections/
<namespace>/<collection>
```

Im Folgenden die möglichen Verzeichnisse und Dateien und ihre Funktionen:

## **README .md**

In der `README .md` werden die oberflächlichen Informationen wie der Einsatzzweck und die Grenzen der Collection beschrieben. Sie soll einen Überblick geben und ist das Erste, was z. B. in einem Git-Repository zu sehen ist.

## **docs**

Das `docs`-Verzeichnis ist als Dokumentationsverzeichnis der Collection gedacht und enthält alle tiefergehenden Informationen, die für den Gebrauch der Collection notwendig sind. Hierzu gehören Details zu den einzelnen Komponenten wie Plugins oder Modul-Dokumentationen.

### **galaxy.yml**

Bei der `galaxy.yml` handelt es sich um die einzige Datei, die in einer Collection zwingend vorhanden sein muss. Sie enthält Informationen über Autor, Lizenzen, Abhängigkeiten und Versionen der Collection. Sofern Sie eine Collection mit dem Programm `ansible-galaxy` erzeugen, wird ein Muster der `galaxy.yml` erstellt, in welchem bereits viele Vorschläge und Kommentare für die erforderlichen Informationen eingefügt sind.

### **plugins**

Im Verzeichnis `plugins` werden alle Plugins und Module der Collection abgelegt. Dabei wird für jedes Plugin ein Unterverzeichnis erstellt und Module werden alle im Unterverzeichnis `plugins/modules` abgelegt. (Auf diese Weise wurden im Kapitel zu Ad-Hoc die Module über Collections installiert.)

### **roles**

Zusätzlich zum Ablegen von Rollen im `/roles`-Verzeichnis wurde mit Collections auch die Möglichkeit eingeführt Rollen in diese zu integrieren. Dies wird möglicherweise zukünftig der Standard werden. Die Rollen behalten dabei ihre gewohnte Verzeichnisstruktur, werden jedoch im Playbook mit Namespace und Collection aufgerufen. Der Aufruf einer Collection in einem Playbook entspricht dem Folgenden:

```
- name: run a role
  hosts: ubuntu1
  roles:
    - namespace.collection.role
```

### **test**

Das `test`-Verzeichnis enthält Test, die für die Übernahme der Collection in die offiziellen Releases erforderlich sind.

### **meta/runtime.yml**

In der `meta/runtime.yml` können Angaben zur Runtime der Collection spezifiziert werden, wie z. B. die exakte `ansible-core`-Version .

### **playbooks**

Das `playbooks`-Verzeichnis bietet die Möglichkeit auch die zur Rolle zugehörigen Playbooks mit in die Collection zu integrieren.

Somit können Playbooks auch in weitere Ansible-Strukturen integriert werden, z. B. über ein

```
import_playbook: namespace.collection.playbook
```

Weitere Informationen über die Verzeichnisse von Collections finden Sie hier:

[https://docs.ansible.com/ansible-devel/dev\\_guide/developing\\_collections\\_structure.html](https://docs.ansible.com/ansible-devel/dev_guide/developing_collections_structure.html)



## Eigene Collections anwenden

- Collection ist in COLLECTION\_PATHS hinterlegt

### Modul aus eigener Collection in task

```
- name: a task to use the custom module
  b1.schulung.test:
    arg: True
```

### Collection in Playbook einbinden:

```
- name: a play with my collection
  hosts: all
  collections:
    - b1.schulung
  tasks:
    - name: a task to use the custom module
      schulungsmodule:
```

Um Collections zu verwenden, die Sie selbst erstellt haben, müssen sich diese in einem Pfad befinden, in dem Ansible nach Collections sucht. Standardmäßig sind dies zwei Pfade

```
# ansible-config dump | grep COLLECTIONS_PATHS
COLLECTIONS_PATHS(default) = ['/root/.ansible/collections',
                             '/usr/share/ansible/collections']
```

nämlich ein Pfad im Heimatverzeichnis des Anwenders und ein globaler Pfad, in dem die Collections systemweit zugänglich sind. Wie bereits im ersten Teil zu Collections zu sehen war, ist es auch möglich, zwei verschiedene Versionen derselben Collection in den zwei Pfaden abzulegen, wobei im Konfliktfall die aus dem Heimatverzeichnis verwendet wird. Wenn Sie mit `ansible-galaxy collection init` eine neue Collection erstellen, prüft `ansible-galaxy` zunächst, ob es den verwendeten Namespace bereits gibt und erzeugt ggf. ein Verzeichnis für diesen. Standardmäßig erzeugt `ansible-galaxy` die Verzeichnisse für Namespace und Collection im jeweiligen Arbeitsverzeichnis. Mit der Option `--init-path <path>` können Sie direkt das gewünschte Verzeichnis definieren, in dem die Collection erzeugt werden soll. Liegt Ihr Collections-Verzeichnis in einem in der Variable `COLLECTIONS_PATHS` eingetragenen Pfad, so können Sie z. B. die darin enthaltenen Module in der gewohnten Art in einem Task verwenden:

```
- name: a task to use the custom module
  b1.schulung.test:
    arg: True
```

In diesem Fall wird ein (fiktives) Modul `test` aufgerufen, welches sich in der Collection `schulung` aus dem Namespace `b1` befindet.

### Collections in Playbooks einbinden

Möchten Sie bei einem Playbook oder einer Rolle nicht bei jedem Task erneut Namespace und Collection dezidiert angeben, so besteht die Möglichkeit, die Collection in das Playbook einzubinden:

```
---
```

```
- name: a play with my collection
  hosts: all
  collections:
    - name: b1.schulung
  tasks:
    - name: a task to use the custom module
      schulungsmodule:
        arg: True
```

In diesem Fall wird über den Eintrag `collections:` die Collection `schulung` unter Angabe des Namespace `b1` geladen. Dies ermöglicht es das Modul `b1.schulung.schulungsmodule` ohne Angabe von Namespace und Collection einfach über den Eintrag `schulungsmodule` zu verwenden. An `collections:` kann auch eine Liste von mehreren Collections übergeben werden.

### Angabe von Namespace und Collection für `ansible.builtin`

Dieses globale Laden von Namespace und Collection geschieht automatisch für die Collection `builtin` aus dem Namespace `ansible`. Dies bedeutet, dass die Angabe von Namespace und Collection für diese Module optional ist:

```
- name: a task with a builtin module
  debug:
    msg: "I dont need no namespace.collection"
```

Es wird allerdings empfohlen auch hier stets das `ansible.builtin.` mit anzugeben um mögliche Unklarheiten zu vermeiden.



#### Hinweis Gleichnamige Module in unterschiedlichen Namespaces

Beim Auftreten von Konflikten mit gleich benannten Module in einer Collection und aus `ansible.builtin` werden immer die `builtin`-Module verwendet, selbst wenn die gleichnamigen Module über den Key `collections:` geladen werden. Würde also eine Collection `b1.schulung`, die das Modul `b1.schulung.test` enthält, über `collections:` geladen und das Modul dann einfach als `ping` in einem Task verwendet, so würde in diesem Fall das Modul `ansible.builtin.test` ausgeführt.



## Collections veröffentlichen

<Namespace>.<Collection>, z. B. b1.schulung

- ① Artefakt (.tar.gz-Datei) erzeugen
- ② Artefakt publizieren

### Erstellen eines Collection-Artefakts mit ansible-galaxy

```
# ansible-galaxy collection build
Created collection for b1.schulung at
/root/.ansible/collections/ansible_collections/
b1/schulung/b1-schulung-1.0.0.tar.gz
```

### Veröffentlichen der Collection mit ansible-galaxy

```
# ansible-galaxy collection publish b1-schulung-1.0.0.tar.gz
Publishing collection artifact '/root/.ansible/collections/
ansible_collections/b1/schulung/b1-schulung-1.0.0.tar.gz'
to default https://galaxy.ansible.com/api/
```

Sie können Ihre selbst erstellten Collections auch in der Ansible Galaxy veröffentlichen. Dafür sind zwei Schritte nötig:

1. Ein Collection Artefakt (.tar.gz-Archiv) erzeugen und
2. das Artefakt publizieren.

Für beide Schritte stellt ansible-galaxy collection die passende Option bereit. Zum Erzeugen des Artefakts steigen Sie in das Verzeichnis Ihrer Collection ab und führen den Befehl ansible-galaxy collection mit der Option build aus:

```
# ansible-galaxy collection build Created collection for b1.schulung
  ↳ at /root/.ansible/collections/ansible_collections/
    ↳ b1/schulung/b1-schulung-1.0.0.tar.gz
```

In der Ausgabe ist zu sehen, dass das Artefakt schulung/b1-schulung-1.0.0.tar.gz im selben Verzeichnis erzeugt wurde.

Um dieses Artefakt zu veröffentlichen ist vorausgesetzt, dass Sie über ein API-Token für die Ansible Galaxy verfügen und diesen korrekt hinterlegt haben. Mehr dazu erfahren Sie unter:

[https://docs.ansible.com/ansible/latest/dev\\_guide/developing\\_collections\\_distributing.html#galaxy-specify-token](https://docs.ansible.com/ansible/latest/dev_guide/developing_collections_distributing.html#galaxy-specify-token)

## 16 Ansible Playbooks und Roles

Ist das Token hinterlegt veröffentlicht Sie Ihr Artefakt mit der Option `publish` unter Angabe des Archivs:

```
# ansible-galaxy collection publish bl-schulung-1.0.0.tar.gz
Publishing collection artifact '/root/.ansible/collections/
    ↳ ansible_collections/bl/schulung/bl-schulung-1.0.0.tar.gz' to
    ↳ default https://galaxy.ansible.com/api/
```

Weitere Informationen über die Entwicklung von Collections und die Partizipation an der Ansible Galaxy erhalten Sie hier:

[https://docs.ansible.com/ansible/latest/dev\\_guide/developing\\_collections.html](https://docs.ansible.com/ansible/latest/dev_guide/developing_collections.html)

### 16.7.1 Aufgabenteil



## Aufgabenteil – Collections 2

- ① Erstellen Sie eine Collection `schulung` im Namespace `b1` im Pfad der Variablen `COLLECTIONS_PATHS`, der sich in Ihrem Heimatverzeichnis befindet.
- ② Kopieren Sie Ihre Rolle `apache2` in das Rollenverzeichnis Ihrer Collection.
- ③ Führen Sie die Rolle aus der Collection heraus aus.

1. Erstellen Sie eine Collection `schulung` im Namespace `b1` im Pfad der Variablen `COLLECTIONS_PATHS`, der sich in Ihrem Heimatverzeichnis befindet.
2. Kopieren Sie Ihre Rolle `apache2` in das Rollenverzeichnis Ihrer Collection.
3. Führen Sie die Rolle aus der Collection heraus aus.

## Musterlösung

1. Erstellen Sie eine Collection **schulung** im Namespace **b1** im Pfad der Variablen **COLLECTIONS\_PATHS**, der sich in Ihrem Heimatverzeichnis befindet:

Sie haben für die Lösung der Aufgabe drei Möglichkeiten:

- a) Erstellen Sie die Collection an einem beliebigen Ort und kopieren Sie diese anschließend nach `~/ ansible/collections/ansible_collections/`.

```
# ansible-galaxy collection init b1.schulung && cp -r b1
  ↪ ~/ansible/collections/ansible\_\_collections/
- Collection b1.schulung was created successfully
```

- b) Wechseln Sie in das Verzeichnis

`~/ ansible/collections/ansible_collections/` und erstellen Sie dann Ihre Collection.

```
# cd ~/ansible/collections/ansible\_\_collections/ &&
  ↪ ansible-galaxy collection init b1.schulung
- Collection b1.schulung was created successfully
```

- c) Übergeben Sie das korrekte Verzeichnis beim Erstellen mit der Option `--init-path`:

```
# ansible-galaxy collection init b1.schulung --init-path
  ↪ ~/ansible/collections/ansible_collections/
- Collection b1.schulung was created successfully
```

2. Kopieren Sie Ihre Rolle **apache2** in das Rollenverzeichnis Ihrer Collection:

Kopieren Sie das Verzeichnis der Rolle aus dem `/root/code/roles`-Verzeichnis in das Rollenverzeichnis der Collection:

```
# cp -r /root/code/roles/apache2
  ↪ ~/ansible/collections/ansible_collections/b1/schulung/roles/
```

3. Führen Sie die Rolle aus der Collection heraus aus:

Ergänzen Sie den Namespace und die Collection in dem Playbook in dem Sie die Rolle `apache2` aufgerufen haben:

```
---
- name: a playbook to include the role apache2
  hosts: all
  roles:
    - b1.schulung.apache2
```

Führen Sie das Playbook wie gewohnt aus:

```
# ansible-playbook -i /root/code/inventory
  ↪ role_apache2_playbook.yml
```

## 17 Ansible Advanced



# Ansible Advanced

## 17.1 Zielsetzung



# Zielsetzung

In diesem Kapitel

- werden die Konfigurationsmöglichkeiten von Ansible gezeigt
- wird Ansible Vault vorgestellt
- werden die komplexen Prozesskontrollwerkzeuge von Ansible erläutert

## 17.2 Ansible – Konfiguration



# Ansible – Konfiguration

- Konfigurationsdatei: `ansible.cfg`
- Einlesereihenfolge:
  - `ANSIBLE_CONFIG`
  - `./ansible.cfg`
  - `~/.ansible.cfg`
  - `/etc/ansible/ansible.cfg`
- globales Inventory standardmäßig in `/etc/ansible/hosts`
- Setzen des Pfades zum Inventory in der `ansible.cfg`

Ausgabe aller Konfigurationsparameter auf der Kommandozeile:

```
# ansible-config dump
```

## Globale Konfigurationsdatei

Ansible kann mit einer globalen Konfigurationsdatei `ansible.cfg` betrieben werden. Standardmäßig wird die Datei im Verzeichnis `/etc/ansible/` angelegt, jedoch mit auskommentierten Werten. Die `ansible.cfg` kann aber an verschiedenen Stellen im System abgelegt werden, die Einlesereihenfolge entspricht dabei:

```
ANSIBLE_CONFIG
./ansible.cfg
~/.ansible.cfg
/etc/ansible/ansible.cfg
```

In der `ansible.cfg` können alle Parameter von Ansible voreingestellt und definiert werden, sodass Ansible vollständig an die jeweilige Umgebung angepasst werden kann.

Eine ausführliche Liste aller Konfigurationsmöglichkeiten finden Sie hier:

[https://docs.ansible.com/ansible/latest/reference\\_appendices/config.html#ansible-configuration-settings](https://docs.ansible.com/ansible/latest/reference_appendices/config.html#ansible-configuration-settings)

### Globales Inventory

Ansible kann ein globales Inventory verwenden, in dem alle standardmäßig zur Landschaft gehörenden Systeme eingetragen werden können. Ist dieses zentrale Inventory erstellt, müssen den Ansible-Befehlen keine Inventories mehr mit der Option –i mitgegeben werden. Das globale Inventory wird in der Datei

```
/etc/ansible/hosts
```

in der gleichen Weise wie sonstige Inventories angelegt.

Alternativ kann auch ein anderer Ort für das globale Inventory festgelegt werden. Dafür setzen Sie den gewünschten Pfad als Wert für die Variable `inventory =` in der `ansible.cfg`:

```
inventory      = ~/code/inventory
```

### Ausgabe der aktuellen Konfiguration

Mit dem Werkzeug `ansible-config` stellt Ansible eine Möglichkeit zur Betrachtung der Konfiguration zur Verfügung. Mit dem Befehl `ansible-config dump` erhalten Sie eine Auflistung aller gesetzten Konfigurationsparameter:

```
# ansible-config dump
ACTION_WARNINGS(default) = True
[...]
ANSIBLE_NOCOWS(default) = False
ANSIBLE_PIPELINING(default) = False
ANY_ERRORS_FATAL(default) = False
BECOME_ALLOW_SAME_USER(default) = False
[...]
```

Diese Variablen können Sie in Ihrer `ansible.cfg` überschreiben und so ändern. Zusätzlich kennt `ansible-config` noch folgende nützliche Optionen:

**list** Ausgabe einer Liste aller möglichen Parameter samt Beschreibung

**view** Ausgabe der aktuell verwendeten `ansible.cfg`

**init** Erzeugen einer neuen Konfigurationsdatei

### 17.2.1 Aufgabenteil



## Aufgabenteil – Konfiguration

- ① Übertragen Sie Ihr Inventory in die Datei /etc/ansible/hosts.
- ② Führen Sie ansible für alle Hosts mit dem Modul ansible.builtin.ping aus.

1. Übertragen Sie Ihr Inventory in die Datei /etc/ansible/hosts.
2. Führen Sie ansible für alle Hosts mit dem Modul ansible.builtin.ping aus.

## Musterlösung

**1. Übertragen Sie Ihr Inventory in die Datei /etc/ansible/hosts:**

```
# cp /root/code/inventory /etc/ansible/hosts
```

**2. Führen Sie ansible für alle Hosts mit dem Modul ansible.builtin.ping aus:**

Geben Sie als Hostdeklaration all und das Modul ansible.builtin.ping mit der Option -m an:

```
# ansible all -m ansible.builtin.ping
ubuntul | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
centos1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
centos2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
suse2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
suse1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
ubuntu2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Ansible nutzt nun standardmäßig das in /etc/ansible/hosts angegebene Inventory.

### 17.3 Ansible Vault



## Ansible Vault

- verschlüsselte Speicherung von Dateien (AES256)
- Passwörter oder Token nicht im Klartext
- Variabldateien in verschlüsselter Form ins VCS
- bestehende Dateien können verschlüsselt werden
- automatische Entschlüsselung bei Ausführung

Bei *Ansible Vault* handelt es sich um ein Werkzeug zur Verschlüsselung sensibler Daten aus Ihrem Ansible Code. Dies bietet sich dann an, wenn mehrere Entwickler an Ihrem Projekt arbeiten, jedoch nicht alle Einblick in die sensiblen Teile erhalten sollen, wie z. B. Passwörter, Verzeichnisse oder Variablen. Diese Vorgehensweise bietet sich ebenfalls an, wenn Sie Ihren Code auf öffentlichen Plattformen wie GitHub oder in Firmen-internen Versionierungstools einsehbar machen möchten.

Ansible Vault ermöglicht es, entweder einzelne Teile oder gesamte Dateien zu verschlüsseln, wodurch gemeinsames Arbeiten am Code möglich bleibt. Mit dem Programm `ansible-vault` können Sie die Dateien verschlüsseln oder einzelne Strings durch AES256-Hashes ersetzen. Ebenso lassen sich Passworddateien und IDs erstellen, um trotz der Verschlüsselung einen automatisierten Ablauf von Ansible zu gewährleisten.

# Ansible Vault: Befehle 1/2

## Verschlüsselte Datei erstellen

```
$ ansible-vault create file.yml
```

## Verschlüsselte Datei bearbeiten

```
$ ansible-vault edit file.yml
```

## Verschlüsselte Datei betrachten

```
$ ansible-vault view file.yml
```

Mit `ansible-vault` können Sie entweder direkt verschlüsselte Dateien erstellen oder bereits vorhandene Dateien nachträglich verschlüsseln. Sie können überdies bereits verschlüsselte Dateien editieren, einsehen, Ihr Passwort ändern oder die Verschlüsselung wieder rückgängig machen.

### Verschlüsselte Dateien erstellen

Mit `ansible-vault` verschlüsselte Dateien erstellen Sie mit dem Parameter `create` wie folgt:

```
# ansible-vault create file.yml
```

Nach Eingabe des Befehls werden Sie aufgefordert, ein Passwort zu vergeben und dieses noch einmal zu bestätigen:

```
# ansible-vault create file.yml
New Vault password:
Confirm New Vault password:
```

Anschließend erhalten Sie eine verschlüsselte Datei:

```
# cat file.yml
$ANSIBLE_VAULT;1.1;AES256
6532386130666230366138656362346361626261653835636135333237333966336332
```

```
34343037646537656334656537363830623636363235356332303965640a6664366563
633735316631663138653838337616532346638393531353530386562626361376636
3065396265326331386337306161363363343336373762330a32383939353930313038
3862646561653935336463373313738346463386463
```

## Bearbeiten von verschlüsselte Dateien

Diese Datei `file.yml` können Sie nun als gewöhnliche YAML-Datei mit Ansible nutzen. Zum Bearbeiten der Datei führen Sie den `ansible-vault`-Befehl mit dem Parameter `edit` aus:

```
# ansible-vault edit file.yml
```

Sie werden nach dem Passwort gefragt und können die Datei anschließend editieren:

```
# ansible-vault edit file.yml
Vault password:
```

## Einsehen von verschlüsselte Dateien

Haben Sie die Datei editiert und geschlossen, können Sie sie mit dem Parameter `view` einsehen:

```
# ansible-vault view file.yml
Vault password:
```

Nach der Passwortheingabe wird Ihnen der Inhalt der Datei angezeigt, in diesem Fall ein Playbook:

```
---
- name: an encrypted playbook
  hosts: suse2
  tasks:
    - name: debug something
      ansible.builtin.debug:
        msg: "I am encrypted and my Os is {{ ansible_distribution }}"
```

## Ansible Vault: Befehle 2/2

### Verschlüsselte Datei entschlüsseln

```
$ ansible-vault decrypt file.yml
```

### Bestehende Datei verschlüsseln

```
$ ansible-vault encrypt \
--output=verschluesselt.yml \
unverschluesselt.yml
```

### Passwort einer bestehenden Datei ändern

```
$ ansible-vault rekey file.yml
```

### Verschlüsselte Dateien entschlüsseln

Um verschlüsselte Dateien wieder zu entschlüsseln, führen Sie `ansible-vault` mit dem Parameter `decrypt` aus:

```
# ansible-vault decrypt file.yml
Vault password:
Decryption successful
```

Nachdem Sie das Passwort eingegeben haben, ist die Datei nicht mehr verschlüsselt:

```
# cat file.yml
---
- name: an encrypted playbook
  hosts: suse2
  tasks:
    - name: debug something
      ansible.builtin.debug:
        msg: "I am encrypted and my Os is {{ ansible_distribution }}"
```

## Bestehende Dateien verschlüsseln

Falls Sie bereits vorhandene Dateien verschlüsseln möchten, so können Sie dies mit dem Parameter `encrypt` tun:

```
# ansible-vault encrypt file.yml
New Vault password:
Confirm New Vault password:
Encryption successful
```

Nach der Eingabe des Passworts ist die Datei nun verschlüsselt:

```
# cat file.yml
$ANSIBLE_VAULT;1.1;AES256
6564666130653231353931623835363338333138366564373863366530393563366233
63356265636261393937313361393566313237376365346230346430350a3336323263
3861643633373832303762643066613066303732373938356563643039343532623130
3034653463663130373965353130333132323162363031380a66653761626235643461
6563383263333066663134323065393930346161636366663332303863663964363330
3764383634383865306364643161663833339335616635666532432393965396663
3932636538623562343535313830353531343266623730356437666566656166663337
316563323734362343561393131303564663135383535643736633537646264303163
3838336534303763653633306131303331323362353432366438666465336362396634
306439646534623835636333233613865646231636433353562643730363665656433
3636646636313830396231313463633866313936616438626461363336343631616432
3634656234326537623631363764316464386537646530623131313931626262633835
6231626633646561303963346432663161396439393431303262633133323466363932
636264333666316331373136343539663234613230636239396362
```

## Passwort einer bestehenden Datei ändern

Das Passwort einer verschlüsselten Datei ändern Sie mit dem Parameter `rekey`:

```
# ansible-vault rekey file.yml
Vault password:
New Vault password:
Confirm New Vault password:
Rekey successful
```

# Passwörter für Ansible Vault

- verschlüsselte Playbooks ausführen
- Zugriff erfordert Passwort
  - Parameter --ask-vault-pass
  - Parameter --vault-password-file
- Dateien werden „entschlüsselt“ im Arbeitsspeicher vorgehalten

## Verschlüsselte Playbooks ausführen

Wenn Sie mit `ansible-vault` ganze Dateien verschlüsseln, kann Ansible diese trotzdem ausführen und benutzen. Zur Nutzung eines verschlüsselten Playbooks gibt es den `ansible-playbook` Parameter `--ask-vault-pass`. Ansible entschlüsselt die Playbook-Datei und führt dieses wie gewohnt aus:

```
# ansible-playbook -i /root/code/inventory --ask-vault-pass file.yml
Vault password:

PLAY [an encrypted playbook] **

TASK [Gathering Facts] **
ok: [suse2]

TASK [debug something] **
ok: [suse2] => {
    "msg": "I am encrypted and my Os is openSUSE Leap"
}

PLAY RECAP **
suse2 : ok=2     changed=0      unreachable=0      failed=0
```

Durch den Parameter `--ask-vault-pass` fragt `ansible-playbook` nach dem Vault-Passwort. Nach der Passworteingabe wird die Playbook-Datei entschlüsselt und das Playbook wie gewohnt abgearbeitet.

### Verschlüsselte Playbooks mit Passworddatei ausführen

Nutzen Sie mehrere mit `ansible-vault` verschlüsselte Komponenten oder arbeiten in einer automatisierten Umgebung, so kann es vorteilhaft sein, das Passwort in eine Datei auszulagern, damit Sie es nicht jedes Mal manuell eingeben müssen. In diesem Fall geben Sie mit dem Parameter `--vault-password-file` eine Datei an, in der Sie das Passwort abgelegt haben.

Hier liegt das `ansible-vault`-Passwort in der Datei `password_file`:

```
# cat password_file
b1s
```

Diese Datei übergeben Sie jetzt als Argument an den Parameter  
`--vault-password-file`:

```
# ansible-playbook -i /root/code/inventory --vault-password-file
    ↪ password_file.yml

PLAY [an encrypted playbook] **

TASK [Gathering Facts] **
ok: [suse2]

TASK [debug something] **
ok: [suse2] => {
    "msg": "I am encrypted and my Os is openSUSE Leap"
}

PLAY RECAP **
suse2 : ok=2     changed=0     unreachable=0     failed=0
```

Durch die Angabe einer Passworddatei können Sie in automatisierten Umgebungen mit Ansible Vault arbeiten.



# Verschlüsseln einzelner Strings

## Verschlüsseln einzelner Strings

```
# ansible-vault encrypt_string 'Value of the Variable' \
--name 'THE_VARIABLE'
New Vault password:
Confirm New Vault password:
THE_VARIABLE: !vault |
$ANSIBLE_VAULT;1.1;AES256
35373737[...]
Encryption successful
```

## Verschlüsselte Variable in einem Playbook

```
vars:
  THE_VARIABLE: !vault |
    $ANSIBLE_VAULT;1.1;AES256
    35373737[...]
```

## Verschlüsseln einzelner Strings

Arbeiten Sie in einer Umgebung gemeinsam mit anderen am Ansible-Code, empfiehlt es sich, nicht die gesamten Dateien, sondern nur die sensiblen Teile zu verschlüsseln. Mit der Option `encrypt_string` können Sie mit `ansible-vault` einzelne Strings verschlüsseln:

```
# ansible-vault encrypt_string 'Value of the Variable' --name
  ↪ 'THE_VARIABLE'
New Vault password:
Confirm New Vault password:
THE_VARIABLE: !vault |
$ANSIBLE_VAULT;1.1;AES256
35373737323536613435313537646534363062666232326433383135326164
33383565386136316134623464666132363665303561343635343530616230
35380a3164666335386433330333135636336303563636537636338656336
63363839303833353339653031636462313161323533613965323265393432
38373137300a64393466363832316163633763663132316232393032323464
65303734633336313938343234373834663930623762353630343131353161
3237613961366235
Encryption successful
```

Der angegebene String wird dabei von `ansible-vault` in einer Variable als verschlüsselter Wert deklariert. Der Aufbau des Befehls ist dabei wie folgt:

```
# ansible-vault encrypt_string 'Value of the Variable' --name
    ↪ 'THE_VARIABLE'
```

Zunächst wird der Option `encrypt_string` der zu verschlüsselnde String übergeben. In diesem Fall ist das „Value of the Variable“. Mit dem Parameter `--name` wird der Name der Variablen festgelegt, in diesem Fall „`THE_VARIABLE`“.

Diese Variable kann dann anstelle der sensiblen Daten, z. B. Login-Daten, angegeben werden. Dabei kann die Variable an allen Orten in Ansible eingesetzt werden, wie andere Variablen auch:

```
# cat vaultvar.yml
---
- name: encrypted variables everywhere
  hosts: ubuntu1
  vars:
    THE_VARIABLE: !vault |
      $ANSIBLE_VAULT;1.1;AES256
      353737373235361343531353764653436306266623232643338313532
      6164333835653861363161346234646661323636653035613436353435
      3061623035380a3164666335386433330333135636336303563636537
      6363386563366336383930383335333965303163646231316132353361
      396532326539343238373137300a643934663638323161636337636631
      323162323930323234646530373463336313938343234373834663930
      6237623536303431313531613237613961366235
  tasks:
    - name: Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de
      ansible.builtin.debug:
        msg: "{{ THE_VARIABLE }}"
```

Das Playbook wird ebenfalls unter der Angabe des Passworts oder einer Passwort-Datei ausgeführt:

```
# ansible-playbook -i /root/code/inventory --vault-password-file
    ↪ password_file vaultvar.yml
```

```
PLAY [encrypted variables everywhere] **

TASK [Gathering Facts] **
ok: [ubuntu1]

TASK [debug] **
ok: [ubuntu1] => {
    "msg": "Value of the Variable"
}

PLAY RECAP **
ubuntu1      : ok=2      changed=0      unreachable=0      failed=0
```

Bis auf den Wert der Variablen `THE_VARIABLE` bleibt das restliche Playbook unverschlüsselt und für jeden lesbar.



### 17.3.1 Aufgabenteil



## Aufgabenteil – Ansible Vault

- ① Verschlüsseln Sie das Playbook `role_apache2_playbook.yml` mit Ansible Vault und führen Sie dieses anschließend aus.
- ② Entschlüsseln Sie das Playbook `role_apache2_playbook.yml` wieder.
- ③ Erstellen Sie eine verschlüsselte Variable `vault_var`. Schreiben Sie ein Playbook für den Host `suse2`, in dem die Variable in einem Task mit dem `ansible.builtin.debug`-Modul aufgerufen und der deklarierte Wert im Klartext ausgegeben wird.

1. Verschlüsseln Sie das Playbook `role_apache2_playbook.yml` mit Ansible Vault und führen Sie dieses anschließend aus.
2. Entschlüsseln Sie das Playbook `role_apache2_playbook.yml` wieder.
3. Erstellen Sie eine verschlüsselte Variable `vault_var`. Schreiben Sie ein Playbook für den Host `suse2`, in dem die Variable in einem Task mit dem `ansible.builtin.debug`-Modul aufgerufen und der deklarierte Wert im Klartext ausgegeben wird.

## Musterlösung

1. Verschlüsseln Sie das Playbook `role_apache2_playbook.yml` mit Ansible Vault und führen Sie dieses anschließend aus:

Verschlüsseln Sie die Playbook-Datei mit `ansible-vault` und der Option `encrypt`:

```
# ansible-vault encrypt role_apache2_playbook.yml
New Vault password:
Confirm New Vault password:
Encryption successful
```

Das Playbook ist nun verschlüsselt:

```
# cat role_apache2_playbook.yml
$ANSIBLE_VAULT;1.1;AES256
633131663862656232656130356263656231663165303337663330663339653663
38626661386461633436346333763366133343736396265613030623361620a31
31306233393131313161333303937383930643962316635353365303635613133
6433663533663639393062383831663134656664633762383865643166310a3730
353238306639343866663061343934313163396164313233646165663863633938
346265623536346366333137653738333761353938333166646431646562326135
643639663431343837323865353632653838653433623264643531393730326133
666130306564656534653637646139313161336566313734663762646366363064
643336333064303931633436333432663164643334623265313930316662656665
34623761653635383562383661336338663466393339633035
```

Führen Sie das Playbook nun mit der Option `--ask-vault-pass` aus:

```
# ansible-playbook --ask-vault-pass role_apache2_playbook.yml
Vault password:

PLAY [a playbook to include the role apache2] **

TASK [Gathering Facts] **
ok: [ubuntul]
ok: [centos2]

TASK [apache2 : include_tasks] **
included: /root/code/roles/apache2/tasks/until.yml for centos2,
    ↪ ubuntul

TASK [apache2 : until this webpage is up] **
skipping: [ubuntul]
ok: [centos2]
[...]
```

**2. Entschlüsseln Sie das Playbook mit dem Befehl `ansible-vault` und der Option `decrypt`:**

```
# ansible-vault decrypt role_apache2_playbook.yml
Vault password:
Decryption successful
```

Anschließend liegt das Playbook wieder im Klartext vor:

```
# cat role_apache2_playbook.yml
---
- name: a playbook to include the role apache2
  hosts: webserver
  roles:
    - apache2
```

**3. Erstellen Sie eine verschlüsselte Variable `vault_var`. Schreiben Sie ein Playbook für den Host `suse2`, in dem die Variable in einem Task mit dem `ansible.builtin.debug`-Modul aufgerufen und der deklarierte Wert im Klartext ausgegeben wird:**

Erstellen Sie zunächst mit dem Befehl `ansible-vault` und der Option `encrypt_string` den verschlüsselten Wert, den Sie später in der Variablen deklarieren:

```
# ansible-vault encrypt_string 'I am totally encrypted' --name
  ↪ 'vault_var'
New Vault password:
Confirm New Vault password:
vault_var: !vault |
$ANSIBLE_VAULT;1.1;AES256
62626137346666383865613032636631643264393434303161616130
39373831613835653037633763666366343263656638663362383266
65656464306166380a3238363838303333161653334326134306561
36616439363066313265356265616230323235633261663335643830
65363763616561656337316262323838350a31336463316132393363
37393632356138633336356431353436313733326633663832623862
6232323963346336316362393562313061653962313666653065
Encryption successful
```

Erstellen Sie nun das Playbook und deklarieren Sie den verschlüsselten String in der Variablen `vault_var`:

```
---
- name: a playbook with an encrypted variable
  hosts: suse2
  vars:
    vault_var: !vault |
$ANSIBLE_VAULT;1.1;AES256
62626137346666383865613032636631643264393434303161616130
39373831613835653037633763666366343263656638663362383266
65656464306166380a3238363838303333161653334326134306561
36616439363066313265356265616230323235633261663335643830
65363763616561656337316262323838350a31336463316132393363
37393632356138633336356431353436313733326633663832623862
```

```
6232323963346336316362393562313061653962313666653065
```

Erstellen Sie anschließend das Playbook mit dem `ansible.builtin.debug`-Modul, das die Variable ausgibt:

```
---
- name: a playbook with an encrypted variable
  hosts: suse2
  vars:
    vault_var: !vault |
      $ANSIBLE_VAULT;1.1;AES256
      6262613734666383865613032636631643264393434303161616130
      3937383161383565303763376366366343263656638663362383266
      65656464306166380a323836383803333161653334326134306561
      36616439363066313265356265616230323235633261663335643830
      65363763616561656337316262323838350a31336463316132393363
      37393632356138633336356431353436313733326633663832623862
      6232323963346336316362393562313061653962313666653065
  tasks:
    - name:
      debug:
        msg: "{{ vault_var }}"
```

Rufen Sie das Playbook unter Angabe der Option `--ask-vault-pass` auf:

```
# ansible-playbook --ask-vault-pass encrypt_var.yml
Vault password:

PLAY [a playbook with an encrypted variable] **

TASK [Gathering Facts] **
ok: [suse2]

TASK [debug] **
ok: [suse2] => {
    "msg": "I am totally encrypted"
}

PLAY RECAP **
suse2 : ok=2     changed=0     unreachable=0     failed=0
```

## 17.4 Delegation



# Delegation

## Key-Deployment mit delegate\_to:

```
---
- name: keydeployment
  hosts: centos1
  tasks:
    - name: create key
      community.crypto.openssh_keypair:
        path: /root/.ssh/id_ed25519_delegate
        type: ed25519
    - name: get the key content to the clipboard
      ansible.builtin.slurp:
        src: /root/.ssh/id_ed25519_delegate.pub
        register: ssh_public_key
    - name: make sure the key is placed on target host
      ansible.posix.authorized_key:
        user: root
        key: "{{ ssh_public_key['content'] | b64decode | trim }}"
        delegate_to: centos2
```

Mit dem Schlüsselwort `delegate_to` weisen Sie einem Task einen anderen Controller zu. Dies bedeutet, dass der Task, an den Sie das `delegate_to` anhängen, nicht auf Ihrem eigentlichen Controller ausgeführt wird, sondern an einen anderen Host delegiert wird. Nützlich ist dies beispielsweise zur Lastverteilung, oder wenn Sie Skripte oder Befehle auf den Zielsystemen selbst ausführen möchten.

Im folgenden Beispiel wird ein SSH-Key mit dem Modul `openssh_keypair` aus der Collection `crypto` des Namespace `community` auf dem unter `hosts:` angegebenen System `centos1` erzeugt. Im zweiten Task wird der Inhalt der Key-Datei `id_ed25519_delegate.pub` mit dem `ansible.builtin.slurp`-Modul ausgelesen und in der Variable `ssh_public_key` registriert. Im letzten Task wird der Inhalt der registrierten Variable durch `Jinja2`-Filter formatiert dem `ansible.builtin.authorized_key`-Modul übergeben, welches diesen in der `~/.ssh/authorized_keys` einträgt. Durch das `delegate_to: centos2` wird der Task jedoch auf dem System `centos2` ausgeführt, sodass dort der Key eingetragen wird.

## 17 Ansible Advanced

```
---
- name: keydeployment
  hosts: centos1
  tasks:
    - name: create key
      community.crypto.openssh_keypair:
        path: /root/.ssh/id_ed25519_delegate
        type: ed25519
    - name: get the key content to the clipboard
      ansible.builtin.slurp:
        src: /root/.ssh/id_ed25519_delegate.pub
        register: ssh_public_key
    - name: make sure the key is placed on target host
      ansible.posix.authorized_key:
        user: root
        key: "{{ ssh_public_key['content'] | b64decode | trim }}"
        delegate_to: centos2
```

Auf centos1 wird ein SSH-Key erzeugt, dessen Public Key vom Controller ausgelesen und in der Variable registriert wird. Im Anschluss wird der Inhalt der registrierten Variable per Delegation auf centos2 eingetragen.

## 17.5 Advanced Inventory



# Advanced Inventory

Fortgeschrittene Techniken im Inventory:

- andere Formate im Inventory:
  - JSON
  - YAML
- dynamische Inventories

Das Inventory von Ansible kann sich in seiner Form den Gegebenheiten und Notwendigkeiten Ihrer Landschaft anpassen. Arbeiten Sie beispielsweise mit einer Landschaft, in der die Systeme nicht statisch bleiben, sondern Hosts verschwinden oder fortwährend neue hinzukommen, so können Sie Ihre Inventories dynamisch generieren lassen. Um Ihren individuellen Anforderungen gerecht zu werden, können Sie das Inventory darüber hinaus auch in anderen Formaten wie z. B. in JSON oder in YAML verfassen.

### 17.5.1 Andere Formate im Inventory



## Andere Formate im Inventory

Inventories in den Formaten:

- YAML
- JSON

### Auszug aus einem Inventory im YAML-Format

```
---
all:
  children:
    ubuntu:
      hosts:
        ubuntu1:
[...]
```

### Erstellen eines YAML-Inventory aus einem INI-Inventory

```
# ansible-inventory -i inventory -y --list
```

Alternativ zur bisher verwendeten INI-Syntax können Sie das Inventory auch in der YAML-Syntax oder als JSON-Datei anlegen.

### Das Inventory im INI-Format

Die Schreibweise im INI-Format lässt das Inventory übersichtlich gestalten:

```
[ubuntu]
ubuntu1
ubuntu2

[centos]
centos1
centos2

[suse]
suse1
suse2

[ubuntususe:children]
ubuntu
```

```
suse

[webserver]
centos2
ubuntul
```

### Das Inventory im YAML-Format

Alternativ zum INI-Format können Sie das Inventory auch in der YAML-Syntax verfassen. Das folgende Inventory entspricht dem oben gezeigten im INI-Format:

```
---
all:
  children:
    ubuntu:
      hosts:
        ubuntul:
        ubuntu2:
    centos:
      hosts:
        centos1:
        centos2:
    suse:
      hosts:
        susel:
        suse2:
    ubuntususe:
      children:
        ubuntu:
        suse:
    webserver:
      hosts:
        centos2:
        ubuntul:
```

Ansible verfügt mit dem Werkzeug `ansible-inventory` über die Möglichkeit, INI-Inventories automatisch in die YAML-Syntax zu übertragen:

```
# ansible-inventory -i inventory -y --list
all:
  children:
    centos:
      hosts:
        centos1: {}
        centos2:
          apache_vhosts: &id001
          - docroot: /var/www/b1.de/
            name: b1.de
          - docroot: /var/www/training.b1.de/
            name: training.b1.de
          httpd_package_state: latest
          jinjalist: &id002
          - the first point is the best
```

## 17 Ansible Advanced

```
- this could be a value
- remeber the field index
- so this is 3, right?
- 4-5
ubuntususe:
  children:
    suse:
      hosts:
        suse1:
          apache2_package_state: latest
        suse2:
          apache2_package_state: latest
      liste:
        - first point
        - second point
        - third point
      testdict:
        detail: is green
        name: first name
        subdict:
          reference: here it is
ubuntu:
  hosts:
    ubuntul:
      apache_vhosts: *id001
      httpd_package_state: latest
      jinjalist: *id002
      links:
        - caption: test
          links:Path to /test/test.de
        - caption: test2
          links: http://test.test.de
      navigation: this is the navigation Var
      testlist:
        - test0
        - test1
        - test2
    ubuntu2: {}
ungrouped: {}
webserver:
  hosts:
    centos2: {}
    ubuntul: {}  
Exemplar von peter.wohlfarth@justiz.thueringen.de
```

Dabei liest ansible-inventory alle für die jeweiligen Hosts oder Gruppen deklarierten Variablen aus und deklariert diese direkt im YAML-Inventory. Mit der Option `-i` wird das INI-Inventory angegeben, die Option `-y` legt das YAML-Format fest und die Option `--list` sorgt für die Ausgabe am Bildschirm. Wird ansible-inventory ohne die Option `-y` ausgeführt, so liefert es JSON.

## Das Inventory im JSON-Format

Neben dem YAML-Format können Sie Ihr Inventory auch in der JSON-Syntax verfassen:

```
{
  "all": {
    "children": {
      "ubuntu": {
        "hosts": {
          "ubuntul": null,
          "ubuntu2": null
        }
      },
      "centos": {
        "hosts": {
          "centos1": null,
          "centos2": null
        }
      },
      "suse": {
        "hosts": {
          "suse1": null,
          "suse2": null
        }
      },
      "ubuntususe": {
        "children": {
          "ubuntu": null,
          "suse": null
        }
      },
      "webserver": {
        "hosts": {
          "centos2": null,
          "ubuntul": null
        }
      }
    }
  }
}
```

Eine Entsprechung des YAML-Inventories samt der Variablen sieht so aus:

```
{  
    "all": {  
        "children": {  
            "centos": {  
                "hosts": {  
                    "centos1": {},  
                    "centos2": {  
                        "apache_vhosts": [  
                            {  
                                "docroot": "/var/www/b1.de/",  
                                "name": "b1.de"  
                            },  
                            {  
                                "docroot": "/var/www/training.b1.de/",  
                                "name": "training.b1.de"  
                            }  
                        ],  
                        "httpd_package_state": "latest",  
                        "jinalist": [  
                            "the first point is the best",  
                            "this could be a value",  
                            "remeber the field index",  
                            "so this is 3, right?",  
                            "4-5"  
                        ]  
                    }  
                }  
            }  
        },  
        "ubuntususe": {  
            "children": {  
                "suse": {  
                    "hosts": {  
                        "suse1": {  
                            "apache2_package_state": "latest"  
                        },  
                        "suse2": {  
                            "apache2_package_state": "latest",  
                            "liste": [  
                                "first point",  
                                "second point",  
                                "third point"  
                            ],  
                            "testdict": {  
                                "detail": "is green",  
                                "name": "first name",  
                                "subdict": {  
                                    "reference": "here it is"  
                                }  
                            }  
                        }  
                    }  
                }  
            }  
        }  
    }  
}
```

Das von `ansible-inventory` aus einem INI-Inventory erzeuge JSON kann von `ansible` nicht als valides Inventory erkannt werden.

## Das Werkzeug ansible-inventory

Das Werkzeug `ansible-inventory` hat neben dem Erstellen von YAML-Versionen aus INI-Inventories noch weitere Funktionen. Mit diesem Werkzeug lassen sich die Inventories und die damit verbundenen Variablen übersichtlich darstellen. Statt der Option `--list` können die Optionen `--graph` und `--host` verwendet werden.

Mit der Option `--graph` können Sie einen grafischen Überblick über alle sich im Inventory befindlichen Hosts und Gruppen erstellen:

```
# ansible-inventory -i inventory --graph
@all:
|--@centos:
| |--centos1
| |--centos2
|--@ubuntususe:
| |--@suse:
| | |--suse1
| | |--suse2
| |--@ubuntu:
| | |--ubuntu1
| | |--ubuntu2
|--@ungrouped:
|--@webserver:
| |--centos2
| |--ubuntu1
```

Mit dem Argument `--vars` für die Option `--graph` können Sie sich zusätzlich die Variablen anzeigen lassen:

```
# ansible-inventory -i inventory --graph --vars
@all:
|--@centos:
| |--centos1
| |--centos2
|--@ubuntususe:
| |--@suse:
| | |--suse1
| | |--suse2
| | |--{apache2_package_state = latest}
| |--@ubuntu:
| | |--ubuntu1
| | |--ubuntu2
|--@ungrouped:
| |--{testvar = I am just a showoff}
|--@webserver:
| |--centos2
| |--ubuntu1
| |--{apache_vhosts = [{u'docroot': u'/var/www/b1.de/', u'name':
| | u'b1.de'}, {u'docroot': u'/var/www/training.b1.de/', u'name':
| | u'training.b1.de'}]}
| |--{httpd_package_state = latest}
```

## 17 Ansible Advanced

```
| --{jinjalist = [u'the first point is the best', u'this could be  
↳ a value', u'remember the field index', u'so this is 3, right?',  
↳ u'4-5']} }
```

Sie können die Auswahl durch Angabe einer Gruppe auch auf diese beschränken:

```
# ansible-inventory -i inventory --graph --vars webserver  
@webserver:  
|--centos2  
|--ubuntu1  
|--{apache_vhosts = [{u'docroot': u'/var/www/b1.de/', u'name':  
↳ u'b1.de'}, {u'docroot': u'/var/www/training.b1.de/', u'name':  
↳ u'training.b1.de'}]}  
|--{httpd_package_state = latest}  
|--{jinjalist = [u'the first point is the best', u'this could be a  
↳ value', u'remember the field index', u'so this is 3, right?',  
↳ u'4-5']} }
```

Mit der Option **--host** und der Angabe eines Hosts können Sie sich die Details für einen einzelnen Host im JSON-Format ausgeben lassen:

```
# ansible-inventory -i inventory --host suse2  
{  
    "apache2_package_state": "latest",  
    "liste": [  
        "first point",  
        "second point",  
        "third point"  
    ],  
    "testdict": {  
        "detail": "is green",  
        "name": "first name",  
        "subdict": {  
            "reference": "here it is"  
        }  
    }  
}
```

Mit der zusätzlichen Option **--yaml** erfolgt die Ausgabe als YAML:

```
# ansible-inventory -i inventory --host suse2 -y  
apache2_package_state: latest  
liste:  
- first point  
- second point  
- third point  
testdict:  
  detail: is green  
  name: first name  
  subdict:  
    reference: here it is
```

### 17.5.2 Dynamische Inventories



## Dynamische Inventories

- Erstellen von dynamischen Inventories aus anderen Quellen wie:
  - LDAP
  - Cobbler
  - OpenStack
  - AWS
  - ...
- Einsatz von Scripten und Plugins

### Dynamische Inventories

Arbeiten Sie in dynamischen Landschaften, in denen öfter Hosts verschwinden oder neue dazukommen, passiert es schnell, dass statisch gepflegte Inventories nicht ausreichen. In diesem Fall können Sie auf dynamische Inventories zurückgreifen und mit Scripten oder Plugins Hosts aus anderen Quellen wie z. B. LDAP, OpenStack, Cobbler oder von Cloud-Anbietern automatisch hinzufügen oder entfernen lassen. Mit *Ansible Tower* existiert ebenfalls eine grafische Lösung zur Verwaltung von dynamischen Inventories.

# Dynamische Inventories mit Plugins

## Ausgabe der verfügbaren Inventory-Plugins

```
# ansible-doc -t inventory -l
```

## Erlauben von Plugins in der ansible.cfg

```
[inventory]
# enable inventory plugins, default: 'host_list', [...]
enable_plugins = host_list, virtualbox, yaml, nmap
```

## Inhalt eines Plugin-Inventory

```
---
plugin: nmap
strict: False
address: 172.20.1.10/24
```

Das Erzeugen dynamischer Inventories ist sowohl mit Scripten als auch mit Plugins möglich. Ansible bringt allerdings von sich aus eine Reihe von Plugins mit und empfiehlt den Einsatz dieser Plugins anstelle selbst geschriebener Scripte.

## **Inventory-Plugins erlauben**

In der Konfigurationsdatei `ansible.cfg` müssen Sie zunächst die gewünschten Plugins im Abschnitt `[inventory]` auf eine Whitelist setzen:

```
[inventory]
# enable inventory plugins, default: 'host_list', 'script', 'yaml',
    ↳ 'ini'
#enable_plugins = host_list, virtualbox, yaml, constructed
```

## **Einsatz eines Inventory-Plugin zum Erstellen eines dynamischen Inventory**

Möchten Sie ein Plugin zur dynamischen Inventory-Erstellung einsetzen, muss dieses zunächst in der `ansible.cfg` aktiviert werden:

```
[inventory]
enable_plugins = nmap
```

Nun ist das Nmap-Plugin von Ansible aktiviert und kann ein dynamisches Inventory aufbauen. Dazu erstellen Sie eine Konfigurationsdatei für das Plugin, im Beispiel nmap\_plugin.yml:

```
---
plugin: nmap
strict: False
address: 172.20.1.10/24
```

Das Nmap-Plugin ermittelt die im Netzwerk auffindbaren Hosts und erzeugt aus diesen das dynamische Inventory.

Geben Sie nun dem Ansible-Befehl die neu erstellte Konfigurationsdatei für das Plugin mit der Option -i als Inventory mit:

```
# ansible all,!controller.training.b1-systems.de -i nmap_plugin.yml
    ↪ -m ansible.builtin.ping
ubuntu1.compose_ansible | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
ubuntu2.compose_ansible | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
centos1.compose_ansible | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
centos2.compose_ansible | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
suse2.compose_ansible | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
suse1.compose_ansible | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Es wird nun für alle im dynamisch erzeugten Inventory eingetragenen Hosts, außer den Controller, das Modul ansible.builtin.ping ausgeführt.

Eine Liste der verfügbaren Inventory-Plugins können Sie sich wie folgt ausgeben lassen:

```
# ansible-doc -t inventory -l
```

Eine Übersicht über Inventory-Plugins finden Sie unter:

<https://docs.ansible.com/ansible/latest/plugins/inventory.html>

## Dynamische Inventories mit Scripten erstellen

Verschiedene Beispiele für Scripte zum Erstellen von dynamischen Inventories mit *Cobbler*, *AWS EC2* und *OpenStack* sowie Informationen zum generellen Erstellen von Scripten für dynamische Inventories finden Sie unter:

[https://docs.ansible.com/ansible/latest/user\\_guide/intro\\_dynamic\\_inventory.html#dynamic-inventory](https://docs.ansible.com/ansible/latest/user_guide/intro_dynamic_inventory.html#dynamic-inventory)

## 17.6 Lookups



# Lookups

- lookup-Plugins: Daten aus anderen Quellen auslesen

### Dynamischer Lookup mit dem Plugin file in einer Variable

```
vars:
  hello: "{{ lookup('file', '/root/code/lookup/hello') }}"
tasks:
  - name: a task to show the looked up var
    ansible.builtin.debug:
      msg: "this file contains: {{ hello }}"
```

### Verfügbare Plugins anzeigen lassen

```
# ansible-doc -t lookup -l
```

Mit `lookup`-Plugins können Sie Daten aus fremden Datenquellen wie z.B. Datenbanken, bestimmten Datentypen oder Containerdateien auslesen und in Ihren Playbooks nutzen.

### **lookup-Plugins benutzen**

Möchten Sie `lookup`-Plugins nutzen, fügen Sie diese mit der folgenden Jinja2-Syntax in Ihre Befehle ein:

```
{{ lookup('<plugin_name>', '<Datenquelle>') }}
```

Nutzen Sie beispielsweise das `file`-Plugin, um eine Datei auszulesen:

```
{{ lookup('file', '/root/code/lookup/hello') }}
```

Der Wert dieser Jinja2-Variable entspricht nun dem Inhalt der Datei `/root/code/lookup/hello`.

In einem debug-Task könnte dies so aussehen:

```
---
- name:
  hosts: ubuntu1
  tasks:
    - name: a task to show the looked up var
      ansible.builtin.debug:
        msg: "{{ lookup('file', '/root/code/lookup/hello') }}"
```

Mit dem Ergebnis:

```
# ansible-playbook lookup.yml

PLAY [ubuntu1] **

TASK [Gathering Facts] **
ok: [ubuntu1]

TASK [debug] **
ok: [ubuntu1] => {
    "msg": "hello ansible!"
}

PLAY RECAP **
ubuntu1 : ok=2      changed=0      unreachable=0      failed=0
```

Eine Liste der lokal verfügbaren lookup-Plugins erhalten Sie mit folgendem Befehl:

```
# ansible-doc -t lookup -l
```

### Installieren zusätzlicher Plugins

Falls Sie zusätzliche Plugins einsetzen möchten, müssen Sie diese in der `ansible.cfg` erlauben und ein Verzeichnis bestimmen, aus dem die Plugins ausgelesen werden sollen:

```
# cat /etc/ansible/ansible.cfg | grep lookup_plugins
#lookup_plugins      = /usr/share/ansible/plugins/lookup
```

Möchten Sie beispielsweise auf Daten aus einem Keypass-Container zugreifen, editieren Sie zunächst Ihre `ansible.cfg` und legen den Pfad fest, in dem Ansible nach lookup-Plugins suchen soll:

```
# cat /etc/ansible/ansible.cfg | grep lookup_plugins
lookup_plugins      = /usr/share/ansible/plugins/lookup
```

Anschließend kopieren Sie die Plugin-Datei in das Verzeichnis:

```
# ls /usr/share/ansible/plugins/lookup
keepass.py
```

Nun wird das Plugin von Ansible gefunden und kann verwendet werden:

```
# ansible-doc -t lookup -l | grep keepass
keepass           fetch data from KeePass file
```

Nachdem das Plugin eingerichtet ist, kann es in Playbooks benutzt werden:

```
---
- name: to look up keepass
  hosts: centos1
  vars:
    keepass_dbx: /root/code/ansible.kdbx
    keepass_psw: b1s
  tasks:
    - name:
      ansible.builtin.debug:
        msg: "The username is{{ lookup('keepass', 'users/Users',
        'username') }} and the password is {{ lookup('keepass',
        'users/Users', 'password') }}"
```

Die Variablen `keepass_dbx` (verwaltet den Pfad zur Keepass-Datei) und `keepass_psw` (verwaltet das Passwort) sollten nach Möglichkeit in den Groupvars oder anderen globalen Orten gesetzt werden. Das unter `keepass_psw` gesetzte Passwort kann und sollte mit `ansible-vault` verschlüsselt werden.

Bei Ausführung liest das `ansible.builtin.debug`-Modul in diesem Playbook den Benutzernamen sowie das Passwort aus dem Eintrag `users/Users` aus:

```
# ansible-playbook keepass.yml

PLAY [to look up keepass] **
  TASK [Gathering Facts] **
ok: [centos1]

  TASK [debug] **
ok: [centos1] => {
    "msg": "The username is:tux and the password is: 12345"
}

PLAY RECAP **
centos1      : ok=2      changed=0      unreachable=0      failed=0
```

Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de

Die Daten könnten so auch direkt an das `user`-Modul übergeben werden, um neue Benutzer anzulegen.

## 17.7 Set Fact



# Set Fact

- Modul `ansible.builtin.set_fact` deklariert Facts dynamisch in einem Playbook
- Facts:
  - setzen einen *Ansible Fact* und keine Variable
  - sind das gesamte Playbook über Plays hinweg gültig

Einbinden des `ansible.builtin.set_fact`-Moduls in einen Task:

```
- name: a task with the set_fact modul
  ansible.builtin.set_fact:
    my_fact: "{{ something otherwise dynamic }}"
```

Mit dem Modul `ansible.builtin.set_fact` können Sie während der Ausführung eines Playbooks dynamisch Facts generieren lassen. Diese Facts werden während des Playbook-Aufrufs deklariert und lassen sich in der Folge im gesamten weiteren Playbook wie gewöhnliche `ansible_facts` verwenden. Im Unterschied zu normalen Variablen werden die Facts, die Sie mit `set_fact` setzen, aus Daten deklariert, die erst mit dem Aufruf des Moduls abgefragt werden. Somit passen sich die so generierten Facts einer sich zwischen den Playbook-Aufrufen ändernden Landschaft an und deklarieren immer die jüngst gegebenen Werte.

Das Modul `ansible.builtin.set_fact` wird wie alle anderen Module auch genutzt:

```
- name: a task with the set_fact modul
  ansible.builtin.set_fact:
    my_fact: "{{ something otherwise dynamic }}"
```

Im Folgenden kann dann der Fact `my_fact` wie eine Variable oder ein Ansible Fact aufgerufen werden.

## Der Unterschied zwischen `set_fact` und Variablen

Es mag auf den ersten Blick nicht direkt einleuchtend sein, welcher Unterschied zwischen Variablen und mit `set_fact` deklarierten Werten besteht. Der Unterschied liegt im Wesentlichen darin, dass die Werte von Variablen bei jedem Aufruf der Variable neu deklariert werden, mit `set_fact` aber nur ein einziges Mal:

```
---
- name:
  hosts: centos1
  vars:
    var_time: "Vartime: {{lookup('pipe', 'date \"+%H:%M:%S\"')}}"
  tasks:
    - name: set time fact
      ansible.builtin.set_fact:
        time: "{{lookup('pipe', 'date \"+%H:%M:%S\"')}}"
    - name: message the time right now
      ansible.builtin.debug:
        msg: "{{ var_time }}"
    - name: message the fact time
      ansible.builtin.debug:
        msg: "{{ time }}"
    - name: sleep for two seconds to see the clock ticking
      ansible.builtin.command: sleep 2
    - name: message the time right now for the second time
      ansible.builtin.debug:
        msg: "{{ var_time }}"
    - name: get the fact time again to compare
      ansible.builtin.debug:
        msg: "{{ time }}"
```

Hier wird der Wert des Befehls `date` einmal in einer Variable `var_time` und einmal im Fact `time` deklariert. Anschließend werden beide Werte mit dem `ansible.builtin.debug`-Modul ausgegeben. Nachdem das Playbook mit dem Befehlt `sleep` im `ansible.builtin.command`-Modul für zwei Sekunden pausiert, wird die Ausgabe der Werte wiederholt.

```
# ansible-playbook set_fact.yml

PLAY [centos1] **

TASK [Gathering Facts] **
ok: [centos1]

TASK [set time fact] **
ok: [centos1]

TASK [message the time right now] **
ok: [centos1] => {
    "msg": "Vartime: 10:34:36"
}

TASK [message the fact time] **
ok: [centos1] => {
```

## 17 Ansible Advanced

```
"msg": "10:34:36"
}

TASK [sleep for two seconds to see the clock ticking] **
changed: [centos1]

TASK [message the time right now for the second time] **
ok: [centos1] => {
    "msg": "Vartime: 10:34:40"
}

TASK [get the fact time again to compare] **
ok: [centos1] => {
    "msg": "10:34:36"
}

PLAY RECAP **
centos1 : ok=7      changed=1      unreachable=0      failed=0
```

Es ist zu erkennen, dass die beiden ersten vom `ansible.builtin.debug`-Modul ausgegebenen Uhrzeiten identisch sind. Nach der Pause hat sich der Wert aus dem Fact im Gegensatz zu dem der Variablen jedoch nicht verändert. Dies liegt daran, dass der Fact mit `set_fact` einmal gesetzt, der Wert der Variablen jedoch bei jedem Aufruf neu deklariert wird.

## 17.8 Magische Variablen



# Magische Variablen

Ansible verfügt über sog. *Magic Variables*:

- sind reserviert
- werden beim Ausführen von Playbooks automatisch deklariert

## Zugriff auf Daten anderer Hosts

```
# ansible suse2 -m ansible.builtin.debug -a "msg={{ hostvars['centos2'] }}"
suse2 | SUCCESS => {
    "msg": {
        "ansible_check_mode": false,
        "ansible_diff_mode": false,
        "ansible_facts": {},
        "ansibleforks": 5,
        "ansible_inventory_sources": [
            [...]
        ]
    }
}
```

Neben den Ansible Facts und normalen Variablen sind in Ansible die sogenannten *Magic Variables* verfügbar. Bei den magischen Variablen handelt es sich um reservierte Variablennamen, für die beim Ausführen von ansible-playbook automatisch generierte Werte deklariert werden. Ansible leitet die Werte dabei aus dem angegebenen Inventory ab. Deklarieren Sie selber Variablen mit Namen aus dieser Liste von Variablen, so werden diese in jedem Fall von Ansible überschrieben.

Sie können magische Variablen genauso einsetzen wie andere Variablen auch.

Im folgenden Beispiel werden einige magische Variablen in der tasks/main.yml einer Rolle eingetragen:

```
---
- name: debug some magic vars
  ansible.builtin.debug:
    msg: "{{ role_path }}"

- name: debug some magic vars
  ansible.builtin.debug:
    msg: "{{ inventory_dir }}"

- name: debug some magic vars
```

## 17 Ansible Advanced

```
ansible.builtin.debug:  
  msg: "{{ playbook_dir }}"  
  
- name: inventory_hostname  
  ansible.builtin.debug:  
    msg: "{{ inventory_hostname }}"
```

Die Rolle wird in einem Playbook eingebunden:

```
---  
- name: some magic vars  
  hosts: webserver  
  roles:  
    - role: b1
```

Und das Playbook anschließend ausgeführt:

```
# ansible-playbook b1role.yml  
  
PLAY [some magic vars] **  
  
TASK [Gathering Facts] **  
ok: [ubuntul]  
ok: [centos2]  
  
TASK [b1 : debug some magic vars] **  
ok: [centos2] => {  
  "msg": "/root/code/roles/b1"  
}  
ok: [ubuntul] => {  
  "msg": "/root/code/roles/b1"  
}  
  
TASK [b1 : debug some magic vars] **  
ok: [centos2] => {  
  "msg": "/etc/ansible"  
}  
ok: [ubuntul] => {  
  "msg": "/etc/ansible"  
}  
  
TASK [b1 : debug some magic vars] **  
ok: [centos2] => {  
  "msg": "/root/code"  
}  
ok: [ubuntul] => {  
  "msg": "/root/code"  
}  
  
TASK [b1 : inventory_hostname] **  
ok: [centos2] => {  
  "msg": "centos2"  
}  
ok: [ubuntul] => {  
  "msg": "ubuntul"
```

}

```
PLAY RECAP **
centos2      : ok=5      changed=0      unreachable=0      failed=0
ubuntul     : ok=5      changed=0      unreachable=0      failed=0
```

Die Werte für die magischen Variablen werden von Ansible automatisch generiert und deklariert, sodass auf diese zugegriffen werden kann. Sie können magische Variablen wie gewohnt nutzen und für Anwendungen, z. B. Templating, beliebig einsetzen. Da magische Variablen von Daten abhängen, die von Ansible mit dem `ansible.builtin.setup`-Modul gesammelt werden, können sie nicht in Ad-Hoc-Befehlen verwendet werden.

### Häufig verwendete magische Variablen

VARIABLE:	BEDEUTUNG:
<code>inventory_hostname</code>	Setzt den Hostnamen aus dem Inventory für den jeweiligen Host ein.
<code>vars</code>	Gibt alle verfügbaren Variablen als Dictionary aus.
<code>hostvars</code>	Gibt die Variablen und Gruppen von Hosts als Dictionary aus.
<code>groups</code>	Gibt ein Dictionary mit den Gruppen und ihren Hosts im verwendeten Inventory aus.
<code>group_names</code>	Zeigt eine Liste aller Gruppen, in denen der jeweilige Host Mitglied ist.
<code>playbook_dir</code>	Enthält als Wert den Pfad zum mit <code>ansible-playbook</code> ausgeführten Playbook.
<code>inventory_dir</code>	Enthält als Wert den Pfad zum verwendeten Inventory.
<code>role_path</code>	Enthält als Wert den Pfad zum Verzeichnis der gerade ausgeführten Rolle.

Eine vollständige Liste aller verfügbaren magischen Variablen können Sie hier einsehen:

[https://docs.ansible.com/ansible/latest/reference\\_appendices/special\\_variables.html#magic](https://docs.ansible.com/ansible/latest/reference_appendices/special_variables.html#magic)



### 17.8.1 Aufgabenteil



## Aufgabenteil – Magische Variablen

Schreiben Sie ein Playbook, in dem Sie in einem Task mit dem `ansible.builtin.debug`-Modul per magischer Variablen die Gruppen ausgeben lassen, in denen das Zielsystem `suse1` eingetragen ist. (Tragen Sie unter `hosts:` nicht `suse1` ein.)

Schreiben Sie ein Playbook, in dem Sie in einem Task mit dem `ansible.builtin.debug`-Modul per magischer Variablen die Gruppen ausgeben lassen, in denen das Zielsystem `suse1` eingetragen ist. (Tragen Sie unter `hosts:` nicht `suse1` ein.)

## Musterlösung

**Schreiben Sie ein Playbook, in dem Sie in einem Task mit dem `ansible.builtin.debug`-Modul per magischer Variablen die Gruppen ausgeben lassen, in denen das Zielsystem `suse1` eingetragen ist. (Tragen Sie unter `hosts:` nicht `suse1` ein):**

Erstellen Sie das Playbook, in dem Sie in dem debug-Task die Hostvars `hostvars['suse1']['group_names']` referenzieren:

```
---
- name: a play to show some hostvars
  hosts: centos1
  tasks:
    - name: a task to message the hostvars for suse1
      ansible.builtin.debug:
        msg: "{{ hostvars['suse1']['group_names'] }}"
```

Wenn Sie das Playbook ausführen, werden Ihnen die Gruppen angezeigt, in denen `suse1` eingetragen ist:

```
# ansible-playbook hostvar.yml

PLAY [centos1] **

TASK [Gathering Facts] **
ok: [centos1]

TASK [debug] **
ok: [centos1] => {
    "msg": [
        "suse",
        "ubuntususe"
    ]
}

PLAY RECAP **
centos1 : ok=2     changed=0     unreachable=0     failed=0
```

## 17.9 Advanced Loops



# Advanced Loops

- nested loop
- until

### Beispiel: Verteilen von Dotfiles verschiedener User

```
- name: Dotfiles verteilen
  ansible.builtin.copy:
    src: "/files/{{ item['0'] }}/{{ item['1'] }}"
    dest: "/home/{{ item['0'] }}/{{ item['1'] }}"
  loop: "{{ ['user1', 'user2']|product(['.vimrc', '.bashrc'])|list }}"
```

### Einen Task bei Fehler wiederholen mit until:

```
- ansible.builtin.get_url:
  url: https://github.com/apache/httpd/archive/trunk.zip
  dest: /opt/software/trunk.zip
  register: download
  until: download['failed'] == false
  retries: 10
  delay: 3
```

Ansible verfügt neben den bisher gezeigten Schleifen noch über weitere, komplexere Typen von Schleifen. Zu diesen gehören Schleifen zum Iterieren über verschachtelte Listen, oder die Until-Schleife.

### Nested Loops

Sie können mit Ansible `loop` über verschachtelte Listen iterieren. Dafür müssen Sie die Listen mit Filtern so aufbereiten, dass sie von `loop` ausgelesen werden können:

```
---
- name: nested loop for the dotfiles
  hosts: centos2
  tasks:
    - name: Dotfiles verteilen
      ansible.builtin.copy:
        src: "/root/code/nested/files/{{ item[0] }}/{{ item[1] }}"
        dest: "/home/{{ item[0] }}/{{ item[1] }}"
      loop: "{{ ['user1', 'user2', 'user3']|product(['.vimrc',
        ↵ '.bashrc'])|list }}"
```

Der Zugriff von `loop` entspricht mit `item[]` dem Zugriff auf bestimmte Punkte einer einzigen Liste. Damit aus zwei Listen eine einzige wird, auf die `loop` zugreifen kann, müssen die beiden Listen mit Filtern so formatiert werden, dass `loop` diese verarbeiten kann. Zunächst kommt dafür der Filter `product()` zum Einsatz. Dieser Filter erzeugt das kartesische Produkt aus iterierbaren Eingabewerten.

Die Funktionsweise lässt sich wie folgt darstellen:

```
product('AB', 'xy')
```

Dies entspricht:

```
((x,y) for x in A for y in B for y in A for x in B)
```

Aus den unter `product()` angegebenen Werten, in diesem Falle (`A, B`), werden mit den Iterationsobjekten, in diesem Falle (`x, y`) in `for`-Schleifen kartesische Produkte, d.h. Paare, gebildet. Diese entsprechen Paaren in allen möglichen Kombinationen der Ausgangswerte:

```
(xA) (yB) (yA) (xB)
```

Aus den unter `loop` angegebenen Listen werden ebensolche kartesischen Produkte gebildet.

Die beiden Listen:

```
{{ ['user1', 'user2', 'user3'] | product(['.vimrc', '.bashrc']) }}
```

werden im Produkt zu folgenden Paaren:

```
(user1, .vimrc) (user1, .bashrc) (user2, .vimrc) (user2, .bashrc)  
  ↔ (user3, .bashrc) (user3, .vimrc)
```

Damit diese Paare von `loop` iteriert werden können, müssen Sie mit dem Filter `list` zu Listen formatiert werden. Es entstehen Listen mit jeweils zwei Einträgen, die dann von `loop` iteriert, und in denen mit dem Feldindex einzelne Listenpunkte referenziert werden können.

Die Listenpaare in YAML-Syntax:

```
['user1', '.vimrc'],  
 ['user1', '.bashrc'],  
 ['user2', '.vimrc'],  
 ['user2', '.bashrc'],  
 ['user3', '.bashrc'],  
 ['user3', '.vimrc']
```

Diese Listen können dann mit folgendem Ergebnis in dem oben gezeigten `loop` iteriert werden:

```
# ansible-playbook nested_dotfiles.yml  
  
PLAY [nested loop for the dotfiles]**  
  
TASK [Gathering Facts] **  
ok: [centos2]
```

```

TASK [Dotfiles verteilen] **
ok: [centos2] => (item=['user1', '.vimrc'])
ok: [centos2] => (item=['user1', '.bashrc'])
ok: [centos2] => (item=['user2', '.vimrc'])
ok: [centos2] => (item=['user2', '.bashrc'])
ok: [centos2] => (item=['user3', '.vimrc'])
ok: [centos2] => (item=[u'user3', u'.bashrc'])

PLAY RECAP ***
centos2 : ok=2     changed=0    unreachable=0    failed=0

```

Durch die Anwendung der Filter `product()` und `list` können also zwei verschachtelte Listen so formatiert werden, dass diese mit `loop` iteriert werden können.

## Until

Mit `until`-Schleifen können Sie einen Task so lange wiederholen lassen, bis eine bestimmte Bedingung eingetroffen ist:

```

---
- hosts: suse2
  tasks:
    - name: make sure git repo is cloned
      ansible.builtin.get_url:
        url: https://github.com/apache/httpd/archive/trunk.zip
        dest: /opt/software/trunk.zip
        register: download
        until: download['failed'] == false
        retries: 10
        delay: 3

```

In diesem Task soll das Modul `get_url` das Zip-Archiv unter dem angegebenen Link herunterladen. Für den Fall, dass der Server oder die Verbindung dorthin instabil sind, wird die `until`-Schleife an den Task angefügt. Diese überprüft, ob das Modul `get_url` die Datei erfolgreich finden und herunterladen konnte. Wenn dies nicht geschehen ist, wird der Task nach 3 Sekunden erneut ausgeführt und die Prüfung wiederholt. Die Zeit, die bis zur Wiederholung gewartet wird, legt das Argument `delay` fest. Mit `retries` wird angegeben, wie oft (im Beispiel also 10 Mal) der Task insgesamt wiederholt werden soll, bevor er als gescheitert markiert wird.

Auszug aus der Until-Schleife:

```

# ansible-playbook until2.yml

PLAY [suse2] **

TASK [Gathering Facts] **
ok: [suse2]

TASK [get_url] **
FAILED - RETRYING: get_url (10 retries left).

```

## 17 Ansible Advanced

```
FAILED - RETRYING: get_url (9 retries left).
FAILED - RETRYING: get_url (8 retries left).
FAILED - RETRYING: get_url (7 retries left).
FAILED - RETRYING: get_url (6 retries left).
[...]
```

### 17.9.1 Aufgabenteil



## Aufgabenteil – Advanced Loops

Erstellen Sie ein Playbook mit einem Task, der auf den Zielsystemen der Gruppe `webserver` Verzeichnisse anlegt. Die Verzeichnispfade sollen sich nach folgendem Schema aus den Listenobjekten der gegebenen Listen zusammensetzen:

```
/var/www/<liste1>/<liste2>
```

Liste 1: ['data', 'storage', 'home', 'IMG', 'box']

Liste 2: ['1', '2', '3', '4', '5', '6']

Erstellen Sie ein Playbook mit einem Task, der auf den Zielsystemen der Gruppe `webserver` Verzeichnisse anlegt. Die Verzeichnispfade sollen sich nach folgendem Schema aus den Listenobjekten der gegebenen Listen zusammensetzen:

```
/var/www/<liste1>/<liste2>
```

LISTE 1:	LISTE 2:
data	1
storage	2
home	3
IMG	4
box	5
	6

## Musterlösung

**Erstellen Sie ein Playbook mit einem Task, der auf den Zielsystemen der Gruppe `webserver` Verzeichnisse anlegt. Die Verzeichnispfade sollen sich nach folgendem Schema aus den Listenobjekten der gegebenen Listen zusammensetzen:**

```
/var/www/<liste1>/<liste2>
```

Erstellen Sie ein Playbook mit einem Play für die Gruppe `webserver`. Dieses enthält einen Task, in dem Sie mit `loop`, dem `product()`-Filter sowie dem `list`-Filter die beiden angegebenen Listen ineinander verschachteln. Lassen Sie das Modul `file` über die Listenobjekte iterieren und die Verzeichnisse anlegen:

```
---
- name: a playbook to implement a nested loop
  hosts: webserver
  tasks:
    - name: create some folders on the targets
      file:
        state: directory
        path: "/var/www/{{ item[0] }}/{{ item[1] }}"
      loop: "{{ ['data','storage', 'home', 'IMG', 'box'] | product(['1','2', '3', '4', '5', '6']) | list }}"
```

Wenn Sie nun das Playbook ausführen, werden die gewünschten Verzeichnisse erstellt:

```
# ansible-playbook nested_exercise.yml

PLAY [a playbook to implement a nested loop] ****
TASK [Gathering Facts] ****
ok: [ubuntu1]
ok: [centos2]

TASK [create some folders on the targets] ****
changed: [ubuntu1] => (item=['data', '1'])
changed: [centos2] => (item=['data', '1'])
changed: [ubuntu1] => (item=['data', '2'])
changed: [centos2] => (item=['data', '2'])
changed: [ubuntu1] => (item=['data', '3'])
changed: [centos2] => (item=['data', '3'])
changed: [ubuntu1] => (item=['data', '4'])
changed: [ubuntu1] => (item=['data', '5'])
changed: [centos2] => (item=['data', '4'])
changed: [ubuntu1] => (item=['data', '6'])
changed: [centos2] => (item=['data', '5'])
changed: [ubuntu1] => (item=['storage', '1'])
changed: [centos2] => (item=['data', '6'])
changed: [ubuntu1] => (item=['storage', '2'])
changed: [ubuntu1] => (item=['storage', '3'])
changed: [centos2] => (item=['storage', '1'])
changed: [ubuntu1] => (item=['storage', '4'])
changed: [centos2] => (item=['storage', '2'])
changed: [ubuntu1] => (item=['storage', '5'])
```

```

changed: [centos2] => (item=['storage', '3'])
changed: [ubuntul] => (item=['storage', '6'])
changed: [centos2] => (item=['storage', '4'])
changed: [ubuntul] => (item=['home', '1'])
changed: [ubuntul] => (item=['home', '2'])
changed: [centos2] => (item=['storage', '5'])
changed: [ubuntul] => (item=['home', '3'])
changed: [centos2] => (item=['storage', '6'])
changed: [ubuntul] => (item=['home', '4'])
changed: [centos2] => (item=['home', '1'])
changed: [ubuntul] => (item=['home', '5'])
changed: [ubuntul] => (item=['home', '6'])
changed: [centos2] => (item=['home', '2'])
changed: [ubuntul] => (item=['IMG', '1'])
changed: [centos2] => (item=['home', '3'])
changed: [ubuntul] => (item=['IMG', '2'])
changed: [centos2] => (item=['home', '4'])
changed: [ubuntul] => (item=['IMG', '3'])
changed: [ubuntul] => (item=['IMG', '4'])
changed: [centos2] => (item=['home', '5'])
changed: [ubuntul] => (item=['IMG', '5'])
changed: [centos2] => (item=['home', '6'])
changed: [ubuntul] => (item=['IMG', '6'])
changed: [centos2] => (item=['IMG', '1'])
changed: [ubuntul] => (item=['box', '1'])
changed: [centos2] => (item=['IMG', '2'])
changed: [ubuntul] => (item=['box', '2'])
changed: [ubuntul] => (item=['box', '3'])
changed: [centos2] => (item=['IMG', '3'])
changed: [ubuntul] => (item=['box', '4'])
changed: [centos2] => (item=['IMG', '4'])
changed: [ubuntul] => (item=['box', '5'])
changed: [centos2] => (item=['IMG', '5'])
changed: [ubuntul] => (item=[u'box', u'6'])
changed: [centos2] => (item=['IMG', '6'])
changed: [centos2] => (item=['box', '1'])
changed: [centos2] => (item=['box', '2'])
changed: [centos2] => (item=['box', '3'])
changed: [centos2] => (item=['box', '4'])
changed: [centos2] => (item=['box', '5'])
changed: [centos2] => (item=['box', '6'])

PLAY RECAP ****
centos2      : ok=2    changed=1    unreachable=0    failed=0
ubuntul     : ok=2    changed=1    unreachable=0    failed=0

```

## 17.10 `include_*` und `import_*`



# `include_*` und `import_*`

Statisches und dynamisches Einfügen in Ansible

Import-Module in Ansible:

- `import_tasks`
- `import_role`
- `import_playbook`

Include-Module in Ansible:

- `include_tasks`
- `include_role`
- `include_vars`

Ansible bietet an vielen Stellen die Möglichkeit, Code-Strukturen modular anzulegen. Wenn Sie an umfangreichen Playbooks und Rollen arbeiten, so müssen Sie nicht alles in eine einzige Datei schreiben, sondern Sie können kleine Abschnitte erstellen und diese anschließend einbinden. So werden z. B. Rollen in Playbooks eingebunden, und innerhalb der Rollenstrukturen werden z. B. einzelne Task-Dateien in die `tasks/main.yml` eingelesen. Ansible unterstützt zwei verschiedenen Arten, Komponenten einzubinden: statisch mit `import_*` oder dynamisch mit `include_*`.

### Statisches Einfügen mit `import_*`

Wenn Sie Inhalte mit dem Modul `import_*` einfügen, so werden diese statisch geladen. Dies bedeutet, dass die Inhalte beim Aufruf des Playbooks eingelesen und geladen werden. Konditionalstatements wie `when:` sowie Tags werden an Tasks weitervererbt, die mit `import_*`-Strukturen geladen werden. Für `import_*` existieren folgende Module:

BEFEHL:	STATISCHES EINBINDEN VON:
import_tasks	Task-Dateien
import_role	Rollen
import_playbook	Playbooks

### Dynamisches Einfügen mit `include_*`

Mit dem Modul `include_*` können Sie Inhalte dynamisch in Ihre Ansible-Strukturen einbinden. Dynamisch bedeutet in diesem Zusammenhang, dass die Inhalte nicht beim Aufruf des Playbooks eingelesen werden, sondern während der Ausführung. Somit können Sie mit Variablen und Schleifen arbeiten. Zu beachten ist, dass Konditionalstatements und Tags nicht an mit `include_*` geladene Tasks weitervererbt werden.

Ein Beispiel für das dynamische Einbinden von Tasks könnte so aussehen:

```
- include_tasks: "{{ ansible_os_family }}.yml
```

Ein Beispiel, in dem Tasks dynamisch mit einer Schleife eingebunden werden, könnte so aussehen:

```
- include_tasks: "{{ item }}.yml
loop:
  - webserver
  - ssh_config
  - dns
```

Für `include_*` existieren folgende Module:

BEFEHL:	DYNAMISCHES EINBINDEN VON:
include_tasks	Task-Dateien
include_role	Rollen
include_vars	Variablen



#### Hinweis Zeitpunkt der Ausführung von eingebundenen Inhalten

Beachten Sie, dass Inhalte, die Sie mit `include_*` oder `import_*` einbinden, an der Stelle im Playbook ausgeführt werden, an der sie aufgerufen werden. Binden Sie eine Rolle mit `include_role` ein, wird diese an der Stelle im Playbook ausgeführt, an der sich der Task befindet, in dem das Modul eingetragen ist. Binden Sie die Rolle mit `roles:` im Playbook ein, wird die Rolle ausgeführt, bevor die im Playbook eingetragenen Tasks ausgeführt werden.

## 17.11 Blocks



# Blocks

- Gruppieren von Tasks
- In-Play Error Handling

## block mit Bedingung

```
- block:
  - name: Debug Message
    ansible.builtin.debug:
      msg: 'User wird erstellt'

  - name: make sure user is present
    ansible.builtin.user:
      name: 'blockuser'
      state: present
  when: ansible_os_family == 'RedHat'
```

Mit dem Parameter `block:` lassen sich beliebig viele Tasks zu einer logischen Einheit zusammenfassen. Dadurch können Konditionalstatements sowie Optionen auf mehrere Tasks auf einmal angewendet werden. Darüber hinaus können innerhalb eines Blocks Maßnahmen definiert werden, die auf das Fehlschlagen von vorherigen Tasks reagieren.

### Mehrere Tasks in einem Block und Vererbung von Optionen

Um Tasks zu einem logischen Block zusammenzufassen, muss dieser mit der Option `block:` als Dictionary eröffnet werden:

```
- block:
  - name: Debug Message
    ansible.builtin.debug:
      msg: 'User wird erstellt'

  - name: make sure user is present
    ansible.builtin.user:
      name: 'blockuser'
      state: present
```

Dieser Block enthält nun zwei Tasks, die zu einer logischen Gruppe zusammengefasst sind. Wird diesem Block z. B. ein when-Konditional angefügt, so wird dieses an jeden einzelnen Tasks innerhalb des Blocks weitervererbt und die Prüfung für jeden Task erneut ausgeführt. In einer Rolle wird Folgendes in der `tasks/main.yml` eingetragen:

```
- block:
  - name: Debug Message
    ansible.builtin.debug:
      msg: 'User wird erstellt'

  - name: make sure user is present
    ansible.builtin.user:
      name: 'blockuser'
      state: present
    when: ansible_os_family == 'RedHat'
```

Die Rolle wird in ein Playbook für alle Hosts eingetragen:

```
# cat blockrole.yml
---
- name: the playbook for the blockrole
  hosts: all
  roles:
    - role: blockrole
```

Und anschließend mit der Option `--limit` für die Systeme `ubuntul` und `centos1` ausgeführt:

```
# ansible-playbook blockrole.yml --limit ubuntul,centos1

PLAY [the playbook for the blockrole] **

TASK [Gathering Facts] **
ok: [ubuntul]
ok: [centos1]

TASK [blockrole : Debug Message] **
skipping: [ubuntul]
ok: [centos1] => {
  "msg": "User wird erstellt"
}

TASK [blockrole : make sure user is present] **
skipping: [ubuntul]
ok: [centos1]

PLAY RECAP **
centos1      : ok=3      changed=0      unreachable=0      failed=0
ubuntul      : ok=1      changed=0      unreachable=0      failed=0
```

Das when-Konditional wurde auf beide Tasks angewendet und die Tasks daher für `ubuntul` übersprungen. Auf diese Art und Weise können beinahe alle Optionen, die auch an einzelne Tasks angefügt werden können, auch an Blocks angefügt werden. Die Optionen werden dann an die einzelnen Tasks weitervererbt. Eine Ausnahme bildet dabei die `loop`-Schleife.

# Blocks: Error Handling

## block Error Handling

```
- block:
  - name: make sure user is present
    ansible.builtin.user:
      name: 'user1'
      state: present
  rescue:
    - name: if something goes south
      ansible.builtin.debug:
        msg: 'something was wrong'
  always:
    - name: always show this message
      ansible.builtin.debug:
        msg: 'this message is always shown'
```

Wenn einer der Tasks in einem Block nicht ausgeführt werden kann, führt dies wie auch bei einzelnen Tasks zum Abbruch des Playbooks. Eine Möglichkeit, den Abbruch des Playbooks zu verhindern, besteht darin, dem Block die Option `ignore_errors: yes` anzuhängen, die dann an alle sich in dem Block befindlichen Tasks weitervererbt wird:

```
---
- name: play to test blocks
  hosts: suse1
  tasks:
    - name: block
      block:
        - ansible.builtin.debug:
            msg: "first message"
        - ansible.builtin.debug:
            msg: " I use a {{ variable }}"
      ignore_errors: yes

- name: just a second play to show the first one!
  hosts: centos1
  tasks:
    - name: do is still run if the first play is kaputt?
      ansible.builtin.debug:
        msg: "this is the second play"
```

In diesem Playbook, das zwei Plays enthält, ist im ersten Play ein Block definiert. In die-

sem Block befinden sich zwei Tasks, wobei der zweite eine nicht deklarierte Variable `variable` aufruft. Dieser Aufruf würde normalerweise das Ausführen des Playbooks an dieser Stelle abbrechen, da die Variable nicht aufgerufen werden kann. Durch die Option `ignore_errors: yes` wird dies jedoch einfach übergangen und das zweite Play wird dennoch ausgeführt:

```
# ansible-playbook block_error.yml

PLAY [play to test blocks] **

TASK [Gathering Facts] **
ok: [suse1]

TASK [debug] **
ok: [suse1] => {
    "msg": "first message"
}

TASK [debug] **
fatal: [suse1]: FAILED! => {"msg": "The task includes an option with
    ↪ an undefined variable. [...]
...ignoring

PLAY [just a second play to show the first one!] **

TASK [Gathering Facts] **
ok: [centos1]

TASK [does this still run if the first play is kaputt?] **
ok: [centos1] => {      Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de
    "msg": "this is the second play"
}

PLAY RECAP **
centos1      : ok=2      changed=0      unreachable=0      failed=0
suse1       : ok=3      changed=0      unreachable=0      failed=0
```

### Die Option `rescue`:

Blocks bieten allerdings eine weitere Möglichkeit, mit Fehlern umzugehen. Mit der Option `rescue:` kann anstelle der Option `ignore_errors: yes` ein weiterer Task erstellt werden, der im Falle eines Fehlers ausgeführt wird. Somit kann für den Fehlerfall nicht nur dafür gesorgt werden, dass das Playbook weiter ausgeführt wird, sondern es können darüber hinaus auch Sofortmaßnahmen eingerichtet werden, die im Fehlerfall greifen:

```
---
- name: play to test blocks
  hosts: suse1
  tasks:
    - name: block
      block:
        - ansible.builtin.debug:
```

## 17 Ansible Advanced

```
        msg: "first message"
- ansible.builtin.debug:
    msg: " I use a {{ variable }}"
rescue:
- name: I am the rescue that does something right up front
  ansible.builtin.debug:
    msg: "Something went wrong here, you might want to have a
         ↪ look?!"
- name: just a second play to show the first one!
hosts: centos1
tasks:
- name: does this still run if the first play is kaputt?
  ansible.builtin.debug:
    msg: "this is the second play"
```

Durch die Option `rescue:` wird das Play fortgesetzt und der dort definierte Task ausgeführt:

```
# ansible-playbook block_rescue.yml

PLAY [play to test blocks] **

TASK [Gathering Facts] **
ok: [suse1]

TASK [debug] **
ok: [suse1] => {
    "msg": "first message"
}

TASK [debug] **
fatal: [suse1]: FAILED! => {"msg": "The task includes an option with
         ↪ an undefined variable. [...]"}

TASK [I am the rescue that does something right up front] **
ok: [suse1] => {
    "msg": "Something went wrong here, you might want to have a look?!"}

PLAY [just a second play to show the first one!] **

TASK [Gathering Facts] **
ok: [centos1]

TASK [does this still run if the first play is kaputt?] **
ok: [centos1] => {
    "msg": "this is the second play"
}

PLAY RECAP **
centos1      : ok=2      changed=0      unreachable=0      failed=0
suse1       : ok=3      changed=0      unreachable=0      failed=1
```

Unter der Option `rescue:` können Sie beliebig viele Tasks definieren, die im Fehlerfall ausgeführt werden. Das Playbook selbst wird anschließend weiter so ausgeführt, als wären alle Tasks in dem Block fehlerfrei durchgelaufen.



### Hinweis *Playbook-Status nach der rescue-Option*

Durch das Anwenden der `rescue:-`-Option wird der Fehlerstatus des Plays innerhalb des Playbooks aufgehoben, nicht jedoch die Meldung, sodass Optionen wie `max_fail_percentage` oder `any_errors_fatal` nicht mehr zuverlässig funktionieren.

## Handlers in Blocks

Wenn Sie innerhalb eines Tasks in einem Block Handler gesetzt haben, so können Sie dafür sorgen, dass auch diese ausgeführt werden, wenn der Task nicht erfolgreich ausgeführt wurde. Neben dem Handler müssen Sie hierfür unter der Option `rescue:` das Modul `ansible.builtin.meta` mit dem Argument `flush_handlers` ausführen:

```
---
- name: a play with a block and a handler
  hosts: suse1
  handlers:
    - name: the handler that'll be used and run after an error
      ansible.builtin.debug:
        msg: "Instead of printing this message I could be doing
              ↪ anything else, restarting a service for example..."
  tasks:
    - name: here comes the block
      block:
        - name: This is the task with the handler
          ansible.builtin.debug:
            msg: "Instead of the debug I could be writing configs..."
            changed_when: yes
          notify: the handler that'll be used and run after an error
        - name: the faulty task with the imagined variable
          ansible.builtin.debug:
            msg: "I call {{ varvar }}"
  rescue:
    - name: here we call the handler with the meta module
      meta: flush_handlers
```

Ohne das `rescue:` würde das Playbook nicht weiter ausgeführt, und da Handler erst am Ende eines Playbooks aufgerufen werden, würden diese nicht mehr „erreicht“. Bei Ausführen eines anderen Moduls unter `rescue:` würde der Handler am Ende des Playbooks wie gewohnt ausgeführt werden. Wenn es allerdings kein anderes Modul gibt, dass Sie unter `rescue:` ausführen wollen, so können Sie das Modul `meta` nutzen, um den Handler aufzurufen:

## 17 Ansible Advanced

```
# ansible-playbook handler_block.yml

PLAY [a play with a block and a handler] **

TASK [Gathering Facts] **
ok: [suse1]

TASK [This is the task with the handler] **
changed: [suse1] => {
    "msg": "Instead of the debug I could be writing configs..."
}

TASK [the faulty task with the imagined variable] **
fatal: [suse1]: FAILED! => {"msg": "The task includes an option with
    ↪ an undefined variable. The error was: 'varvar' is
    ↪ undefined\n\nThe error appears to have been in
    ↪ '/root/code/blocks/handler_block.yml': line 17, column 11, but
    ↪ may\nbe elsewhere in the file depending on the exact syntax
    ↪ problem.\n\nThe offending line appears to be:\n\n
    ↪ name: the faulty task with the imagined variable\n
    ↪ here\n"}
```

RUNNING HANDLER [the handler that'll be used and run after an error] \*\*  
ok: [suse1] => {  
 "msg": "Instead of printing this message I could be doing anything  
 ↪ else, restarting a service for example..."  
}

PLAY RECAP \*\*  
suse1 : ok=3 changed=1 unreachable=0 failed=1

### Die Option `always`:

Zusätzlich zur Option `rescue`: gibt es die Option `always`:, die in der gleichen Art an Blöcke angefügt werden kann:

```
---
- name: play to test blocks
  hosts: suse1
  tasks:
    - name: block
      block:
        - ansible.builtin.debug:
            msg: "first message"
        - ansible.builtin.debug:
            msg: " I use a {{ variable }}"
    always:
      - name: I am always executed, no matter what!
        ansible.builtin.debug:
          msg: "Something went wrong here, or not, doens't matter?!"
```

- name: just a second play to show the first one!

```
hosts: centos1
tasks:
  - name: does this still run if the first play is kaputt?
    ansible.builtin.debug:
      msg: "this is the second play"
```

Unter `always:` definierte Tasks werden immer ausgeführt, unabhängig davon, ob die Tasks in dem Block erfolgreich ausgeführt wurden oder nicht. Ist ein Task in dem Block fehlerhaft, weil z. B. wie hier die Variable `variable` nicht deklariert ist, so wird der Task unter `always:` trotzdem noch vor Abbruch der Ausführung des Playbooks ausgeführt:

```
# ansible-playbook block_rescue.yml

PLAY [play to test blocks] **

TASK [Gathering Facts] **
ok: [suse1]

TASK [debug] **
ok: [suse1] => {
    "msg": "first message"
}

TASK [debug] **
fatal: [suse1]: FAILED! => {"msg": "The task includes an option with
    ↪ an undefined variable. [...]"}

TASK [I am always executed, no matter what!] **
ok: [suse1] => {
    "msg": "Something went wrong here, or not, doens't matter?!"}

PLAY RECAP **
suse1 : ok=3    changed=0    unreachable=0    failed=1
```

Sind alle Tasks in dem Block fehlerfrei, so werden die unter `always:` definierten Tasks auch ausgeführt. Die Optionen `rescue:` und `always:` können auch miteinander kombiniert an einen Block angefügt werden:

```
- name: block
  block:
    - ansible.builtin.debug:
        msg: "first message"
    - ansible.builtin.debug:
        msg: " I use a {{ variable }}"
  rescue:
    - ansible.builtin.debug:
        msg: "In case of error this is shown."
  always:
    - name: I am always executed, no matter what!
      ansible.builtin.debug:
        msg: "Something went wrong here, or not, doens't matter?!"
```

## Schleifen und Blocks

Möchten Sie Schleifen in der Kombination mit Blocks einsetzen, so müssen Sie die Schleifen den einzelnen Tasks anhängen und nicht dem Block:

```
---
- name: play with blocks and loops
  hosts: centos
  tasks:
    - name:
      block:
        - ansible.builtin.debug:
            msg: "{{ item }}"
        loop:
          - '0'
          - '1'
          - '2'
        - ansible.builtin.debug:
            msg: "Yet another message"
```

Die Schleife wird jeweils für den einzelnen Task ausgeführt. Eine Schleife für den gesamten Block zu setzen ist nicht möglich.

## 17.12 Privilege Escalation



# Privilege Escalation

- nicht alle Tasks als root
- manche Tasks als Service User
- kein root-Login erlaubt
- „become“ um auf einen anderen User zu wechseln
- entweder unprivilegiert → root oder root → unprivilegiert

## Passworteingabe

```
$ ansible-playbook --ask-become-pass test.yml
```

## Achtung!

Es kann nur ein Passwort für den Benutzerwechsel mitgegeben werden. Muss auf verschiedene Nutzer gewechselt werden, ist dies nur mit dem root-User möglich.

*Privilege Escalation* dient in erster Linie der Kontrolle der Rechte, mit denen Ansible in der Umgebung seine Befehle ausführt. Hierbei sind drei Szenarien vorstellbar:

1. Die Befehle sollen als privilegierter User ausgeführt werden.
2. Die Befehle sollen als unprivilegierter User ausgeführt werden.
3. Die Befehle sollen als spezieller User (z. B. www-data) ausgeführt werden.

## Privilege Escalation in Ad-Hoc Befehlen

Bei der Anwendung von Ansible müssen Sie unter Umständen mit Superuser-Rechten arbeiten, da Ihnen sonst auf den Zielsystemen die Rechte zum Ausführen von Befehlen fehlen. Möchten Sie beispielsweise Pakete auf einem Zielsystem installieren, so muss das Modul `ansible.builtin.package` auf dem Zielsystem von einem privilegierten Nutzer ausgeführt werden. Ansible sieht hierfür die folgenden Optionen in der Benutzung von Ad-Hoc Befehlen vor:

OPTION:	BEDEUTUNG:
--ask-become-pass   -K	Fordert die Eingabe des Passworts für den privilegierten Nutzer.
--become   -b	Erlaubt das Ausführen von Ansible mit Privilege Escalation.
--become-method	Legt die Art der Privilege Escalation fest; Standard ist sudo (su, ksu, pbrun, ...).
--become-user	Legt den User fest, mit dem das Modul ausgeführt werden soll.

Wenn Sie `ansible` mit privilegierten Rechten auf dem Zielsystem ausführen möchten, gibt es mehrere Möglichkeiten.

Sie können, sofern `sudo` auf dem Zielsystem vorhanden und für Ihren Benutzer verfügbar ist, `ansible` den Befehl mit Ihrem User und `sudo` ausführen lassen:

```
[boringuser@controller ~]$ ansible ubuntul --become --ask-become-pass
    ↪ -m ansible.builtin.package -a "name=nano state=present use=apt"
SUDO password:
```

Zum Ausführen von `ansible` auf dem Zielsystem mit `sudo` müssen Sie die Optionen `--become` und `--ask-become-pass` nutzen. So führen Sie das Modul mit privilegierten Rechten aus und lassen sich von Ansible auffordern, das für `--become` nötige Passwort einzugeben. Sie müssen keine besondere Methode angeben, da `sudo` die Standardeinstellung von `--become` ist. (Haben Sie keinen SSH-Key auf dem Zielsystem platziert, müssen Sie zusätzlich die Option `-k` angeben, um nach dem SSH-Passwort des sich verbindenden Users gefragt zu werden.)

Sie können `ansible` direkt den User `root` auf dem Zielsystem nutzen lassen:

```
[boringuser@controller ~]$ ansible ubuntul --become --become-user root
    ↪ --ask-become-pass --become-method su -m ansible.builtin.package
    ↪ -a "name=nano state=present use=apt"
SU password:
```

Damit Sie `ansible` direkt mit dem Root-User ausführen können, müssen Sie zunächst mit der Option `--become` angeben, dass Sie den Befehl mit privilegierten Rechten ausführen wollen. Anschließend müssen Sie mit der Option `--become-user` angeben, als welcher User der Befehl ausgeführt werden soll. Mit der Option `--ask-become-pass` werden Sie aufgefordert, das Passwort des Nutzers einzugeben, dessen privilegierten Status Sie nutzen möchten, in diesem Falle das Passwort des Users Root. Schließlich müssen Sie in diesem Fall noch die Art der Privilege Escalation festlegen, da der Befehl vom User Root und nicht vom Ansible-User mit Sudo-Rechten ausgeführt werden soll. Diese legen Sie mit `--become-method` fest.

Wenn der Fall besteht, dass Sie ausschließlich über einen privilegierten Zugang verfügen, `ansible` jedoch mit einem unprivilegierten User ausführen lassen wollen, so können Sie dies wie folgt tun:

```
# ansible ubuntul --become --become-user boringuser -m
    ↪ ansible.builtin.shell -a "whoami"
ubuntul | CHANGED | rc=0 >>
boringuser
```

Mit der Option `--become` erlauben Sie das Wechseln des Nutzers mit `--become-user`, der dann das Modul `shell` auf dem Zielsystem ausführt.

Das Ausführen von ansible Ad-Hoc Befehlen mit Systembenutzern wie `www-data` ist *nicht* möglich.



### Hinweis Passworteingabe mit `--ask-become-pass`

Bei Verwendung dieser Option wird Ansible versuchen, dieses Passwort auf allen angegebenen Systemen zu nutzen.

## Privilege Escalation in Playbooks

Playbooks erlauben es, die Privilege Escalation für einzelne Tasks oder für Blöcke festzulegen, sodass verschiedene Tasks innerhalb eines Playbooks mit verschiedenen Rechten bzw. von verschiedenen Usern ausgeführt werden können. Hierfür gibt es Direktiven, die am Ende eines Tasks oder Blocks angefügt werden können:

DIREKTIVE:	BEDEUTUNG:
<code>become</code>	Erlaubt mit dem Argument <code>yes</code> die Privilege Escalation.
<code>become_method</code>	Legt die Art der Privilege Escalation fest; Standard ist <code>sudo</code> ( <code>su</code> , <code>ksu</code> , <code>pbrun</code> , ...).
<code>become_user</code>	Legt den User fest, mit dem das Modul ausgeführt werden soll.
<code>become_flags</code>	Erlaubt die Festlegung spezifischer Flags für den Task oder die Rolle. Ein Beispiel wäre die Angabe einer Shell für Systemuser wie <code>www-data</code> mit „ <code>-s /bin/sh</code> “.

Auch für Playbooks gibt es wieder verschiedene Möglichkeiten der Privilege Escalation. Sie können den Task von einem unprivilegierten Benutzer mit `sudo` ausführen lassen:

```
---
- name: well, that escalated quickly
  hosts: ubuntu
  vars:
  tasks:
    - name: a file task to check someones privileges
      ansible.builtin.file:
        state: touch
        dest: /home/boringuser/showcase.txt
      become: yes
      become_user: boringuser
```

In diesem Fall wird der Task mit dem User `boringuser` ausgeführt, der dann in seinem Homeverzeichnis eine Datei anlegt. Die Direktive `become` ist dabei obligatorisch, um die Privilege Escalation überhaupt zu erlauben. Ohne explizite Angabe einer Methode mit `become_method` wird standardmäßig `sudo` verwendet.

Sie können mit `become_method` eine andere Methode festlegen, mit der die Privilege Escalation erfolgen soll. Dies kann z. B. nötig sein, wenn auf dem Zielsystem kein `sudo` installiert ist:

```
---
- name: well, that escalated quickly
  hosts: ubuntu1
  vars:
  tasks:
    - name: a file task to check someones privileges
      ansible.builtin.file:
        state: touch
        dest: /home/boringuser/showcase.txt
      become: yes
      become_method: su
      become_user: root
```

In diesem Fall wird der Task mit der Methode `su` vom User `root` ausgeführt.

Wenn Sie einen Task als Systembenutzer, z. B. `www-data`, ausführen lassen wollen, so müssen Sie mit `become_flags` noch die nötigen Flags setzen:

```
---
- name: well, that escalated quickly
  hosts: ubuntu1
  vars:
  tasks:
    - name: a file task to check someones privileges
      ansible.builtin.file:
        state: touch
        dest: /var/www/showcase.txt
      become: yes
      become_method: su
      become_user: www-data
      become_flags: -s /bin/sh
```

Mit der Flag `-s /bin/bash` wird dem Systemuser hier eine Shell zugewiesen, in der er den Befehl ausführen kann.

### Verbindungsvariablen für Privilege Escalation

Sie können die Privilege Escalation auch über Variablen erreichen. Diese können entweder im Playbook, in den Host- und Groupvars oder im Inventory eingesetzt werden. Ansible verfügt hierfür über vordeklarierte Variablen:

VARIABLE:	BEDEUTUNG:
ansible_become	Erlaubt mit dem Argument yes die Privilege Escalation.
ansible_become_method	Legt die Art der Privilege Escalation fest; Standard ist sudo (su, ksu, pbrun, ...).
ansible_become_user	Legt den User fest, mit dem das Modul ausgeführt werden soll.
ansible_become_pass	Setzt das Passwort für die Privilege Escalation; kann mit Ansible Vault kombiniert werden.

Obwohl Sie diese Variablen im Playbook oder in den Host- und Groupvars benutzen können, bieten sie sich hauptsächlich an, um im Inventory eingesetzt zu werden. Sie können sie dort als Verbindungsvariablen für einzelne Hosts oder für Gruppen deklarieren:

```
[ubuntu]
ubuntul
ubuntu2 ansible_become=yes ansible_become_user=boringuser

[centos]
centos1
centos2

[suse]
suse1
suse2

[ubuntususe:children]
ubuntu
suse

[webserver]
centos2
ubuntul

[webserver:vars]
ansible_become=yes
ansible_become_user=root
ansible_become_pass='{{ webserver_pass }}'
```

Hier wurden für den Host ubuntul die Variablen `ansible_become` und `ansible_become_user` deklariert, sodass alle Ansible-Befehle auf diesem Host bei Verwendung dieses Inventories als boringuser ausgeführt werden. Für die Gruppe `webserver` wurden Variablen deklariert, sodass alle Befehle mit dem User `root` ausgeführt werden. Zusätzlich wurde mit `ansible_become_pass` das Passwort angegeben. Hinter der Variable `webserver_pass` verbirgt sich ein Vault-Passwort, das in der `group_vars/webserver.yml` deklariert wurde:

```
[boringuser@controller ~]$ cat group_vars/webserver.yml
---
webserver_pass: !vault |
$ANSIBLE_VAULT;1.1;AES256
3431313634626164623831623463616131338363561663464383066306462
39333461393966383265353636663164636264343334303231383466336432
```

## 17 Ansible Advanced

```
34330a35323731656366626637383739636261396533373630303266663366
38623764636463383135346136643233633165656430396363363466386333
65326239300a33303038313961393537613131346638646561366462643634
37366431623536
```

Die Ausführung eines Befehls für die Gruppe webserver erfolgt nun als Benutzer root:

```
[boringuser@controller ~]$ ansible webserver -i inventory -m command
  ↵ -a "whoami" --ask-vault-pass
Vault password:
ubuntul | CHANGED | rc=0 >>
root

centos2 | CHANGED | rc=0 >>
root
```

Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de

## 17.13 Ansible Tagging



# Tagging

- bietet die Möglichkeit, nur Teile von Playbooks und Plays gezielt auszuführen
- Beispiel: nach aktualisierter Paketliste Installation anstoßen
- Format: `tags: <tagname>`
- ein Tagname kann mehrmals verwendet werden
- vordefinierte Tags: `always`, `tagged`, `untagged` und `all`
- `--tags`, `--skip-tags`
- `--list-tags`

Mit Tags können Sie Tasks in Playbooks oder Rollen kennzeichnen. Diese Kennzeichnung erlaubt es Ihnen, das Playbook unter Angabe von Tags auszuführen, wodurch nur die gekennzeichneten Tasks ausgewählt und ausgeführt werden. Wenn Sie ein Playbook entwickeln oder Änderungen an einzelnen Bereichen, wie z. B. Rollen, vornehmen und diese Änderungen überprüfen wollen, so müssen Sie nicht das ganze Playbook durchlaufen lassen, sondern können mit Tags die zu prüfenden Bereiche auswählen.

### Einzelne Tags vergeben und aufrufen

Sie können Tags vergeben, indem Sie diese mit der Option `tags:` an einen Task anfügen. Hier wird der zweite Task mit dem Tag `testtag` gekennzeichnet:

```
# cat tagbook.yml
---
- name: a playbook to tag around
  hosts: centos1
  tasks:
    - name: first task - untagged
      ansible.builtin.debug:
```

```
msg: "I am not tagged"

- name: a task with a tag
  ansible.builtin.debug:
    msg: "This task is tagged"
  tags: 'testtag'
```

Bei Ausführung des das den Task enthaltenden Playbooks mit der Option `--tags` und dem Tag als Argument wird ausschließlich der Task mit dem Tag ausgeführt:

```
# ansible-playbook tagbook.yml --tags "testtag"

PLAY [a playbook to tag around] **

TASK [Gathering Facts] **
ok: [centos1]

TASK [a task with a tag] **
ok: [centos1] => {
    "msg": "This task is tagged"
}

PLAY RECAP **
centos1 : ok=2     changed=0     unreachable=0     failed=0
```

### Mehrere Tags vergeben und aufrufen

Sie können auch mehrere Tags innerhalb eines Playbooks vergeben und aufrufen. Zum einen können Sie Tags an mehrere Tasks vergeben:

```
# cat tagbook.yml
---
- name: a playbook to tag around
  hosts: centos1
  tasks:

    - name: first task with
      ansible.builtin.debug:
        msg: "I am tagged"
      tags: 'firsttag'

    - name: a task with a tag
      ansible.builtin.debug:
        msg: "This task is also tagged"
      tags: 'secondtag'
```

Und beim Ausführen des Playbooks zwei Argumente mit Komma getrennt an die Option `--tags` übergeben:

```
# ansible-playbook tagbook.yml --tags "firsttag,secondtag"

PLAY [a playbook to tag around] **
```

```

TASK [Gathering Facts] **
ok: [centos1]

TASK [first task with] **
ok: [centos1] => {
    "msg": "I am tagged"
}

TASK [a task with a tag] **
ok: [centos1] => {
    "msg": "This task is also tagged"
}

PLAY RECAP **
centos1 : ok=3     changed=0     unreachable=0     failed=0

```

Zum anderen können Sie einzelnen Tasks mehrere unterschiedliche Tags zuweisen. Diese werden dann als YAML-Liste angegeben:

```

---
- name: a playbook to tag around
  hosts: centos1
  tasks:
    - name: first task with
      ansible.builtin.debug:
        msg: "I am tagged"
      tags: [ 'firsttag', 'anothertag' ]

    - name: a task with a tag
      ansible.builtin.debug:
        msg: "This task is also tagged"
      tags: 'secondtag'

```

Der erste Task kann nun mit dem einen oder dem anderen Tag adressiert werden:

```

# ansible-playbook tagbook.yml --tags "firsttag"

PLAY [a playbook to tag around] **

TASK [Gathering Facts] **
ok: [centos1]

TASK [first task with] **
ok: [centos1] => {
    "msg": "I am tagged"
}

PLAY RECAP **
centos1 : ok=2     changed=0     unreachable=0     failed=0

# ansible-playbook tagbook.yml --tags "anothertag"

PLAY [a playbook to tag around] **

```

## 17 Ansible Advanced

```
TASK [Gathering Facts] **
ok: [centos1]

TASK [first task with] **
ok: [centos1] => {
    "msg": "I am tagged"
}

PLAY RECAP **
centos1 : ok=2     changed=0     unreachable=0     failed=0
```

### Tags wiederverwenden

Einzelne Tags können auch mehrfach vergeben und wiederverwendet werden:

```
---
- name: a playbook to tag around
  hosts: centos1
  tasks:
    - name: first task with
      ansible.builtin.debug:
        msg: "I am tagged"
      tags: [ 'firsttag', 'anothertag' ]

    - name: a task with a tag
      ansible.builtin.debug:
        msg: "This task is also tagged"
      tags: 'secondtag'

    - name: yet another task
      ansible.builtin.debug:
        msg: "This is a third task with a tag"
      tags: 'anothertag'
```

Nun können Sie den ersten und den dritten Task mit dem Tag „anothertag“ adressieren:

```
# ansible-playbook tagbook.yml --tags "anothertag"

PLAY [a playbook to tag around] **

TASK [Gathering Facts] **
ok: [centos1]

TASK [first task with] **
ok: [centos1] => {
    "msg": "I am tagged"
}

TASK [yet another task] **
ok: [centos1] => {
    "msg": "This is a third task with a tag"
}
```

```
PLAY RECAP **  
centos1 : ok=3     changed=0     unreachable=0     failed=0
```

Sie können beliebig viele Tags vergeben und somit das Ausführen bestimmter Tasks kombinieren.

### Tags überspringen

Mit der Option `--skip-tags` können Sie Tags überspringen. Dazu muss zusätzlich die Option `--tags` mit dem Argument „all“ aufgerufen werden:

```
# ansible-playbook tagbook.yml --tags "all" --skip-tags "anothertag"
```

```
PLAY [a playbook to tag around] **  
  
TASK [Gathering Facts] **  
ok: [centos1]  
  
TASK [a task with a tag] **  
ok: [centos1] => {  
    "msg": "This task is also tagged"  
}  
  
PLAY RECAP **  
centos1 : ok=2     changed=0     unreachable=0     failed=0
```

# Vererbung von Tags

## Rolle mit Tags

```
---
- name: a play with a role an a tag
  hosts: webserver
  roles:
    - role: httpd
      tags: [ 'webserver', 'webapp' ]
```

## Task mit dem Modul import\_role

```
---
- name: a task to import a role
  import_role:
    name: httpd
  when: ansible_os_family == Debian
  tags: [ 'webserver', 'webapp' ]
```

## Tags in Rollen vererben

Wenn Sie Rollen mit Tags kennzeichnen, so werden die Tags automatisch an alle sich in der Rolle befindlichen Tasks – und nur an diese – vererbt. Die Tags werden in der Abhängigkeitskette der Rolle abwärts vererbt. Möchten Sie, dass die Tags an alle Tasks der Rolle vererbt werden, sollten Sie die Tags vergeben, wenn die Rolle mit `roles:` oder statisch mit `import_role:` geladen wird. In diesem Fall sollten die Tags *nicht* an die einzelnen Tasks innerhalb der Rolle vergeben werden.

Um Tags an eine Rolle zu vergeben, sodass diese an die Tasks der Rolle vererbt werden, können Sie die Rolle direkt beim Einbinden in das Playbook kennzeichnen. Hierfür wird die Rolle entweder mit `roles:` eingebunden:

```
roles: Erster Punkt der Aufgabe.
  - role: httpd
    tags: [ 'webserver', 'webapp' ]
```

oder statisch mit dem Modul `import_role::`:

```
- import_role: httpd
  tags: [ 'webserver', 'webapp' ]
```

In beiden Fällen werden die Tags an die Tasks der Rolle vererbt. Bei der Angabe der Tags beim Ausführen der Rolle werden nur die Tasks dieser Rolle ausgeführt.

Nach dem gleichen Prinzip werden Tags an Tasks vererbt, die mit dem Modul `import_tasks: importiert` werden:

```
- import_tasks: httpd.yml
  tags: [ 'webserver', 'webapp' ]
```



### **Hinweis** `include_tasks:` und `include_roles:`

Tags werden bei Anwendung mit den dynamischen Modulen `include_tasks:` und `include_roles:` nicht an die eingefügten Tasks mitvererbt.

## **Tags in Blocks vererben**

Wenn Sie Tags zusammen mit Blocks einsetzen, so werden die Tags, welche an einem Block gesetzt werden, an jeden einzelnen Task in dem Block vererbt. So werden bei Angabe des Tags alle Tasks des Blocks ausgeführt.

```
- block:
  - name: Debug Message
    ansible.builtin.debug:
      msg: 'User wird erstellt'

  - name: create user
    ansible.builtin.user:
      name: 'user1'
      state: present
  when: ansible_os_family == 'RedHat'
  tags: ['users', 'deploy']
```

# Vordefinierte Tags

Ansible verfügt über einige vordefinierte Tags:

- always
- never

Argumente für --tags:

- tagged
- untagged
- all

## Task mit dem Tag never

```
---
```

```
- name: a tagged task
  ansible.builtin.package:
    name: "{{ httpd_package }}"
  tags: [ 'never', 'webapp' ]
```

## Vordefinierte Tags

Ansible verfügt über eine Reihe vordefinierter Tags, mit denen sich der Aufruf von vergebenen Tags beeinflussen lässt.

Mit dem Tag `always` können Sie dafür sorgen, dass ein Task immer aufgerufen wird, egal welcher Tag mit `--tags` angegeben wurde:

```
---
```

```
- name: a playbook to tag around
  hosts: centos1
  tasks:
    - name: first task with
      ansible.builtin.debug:
        msg: "I am tagged"
      tags: [ 'firsttag', 'anothertag' ]
    - name: this task has an always tag
      ansible.builtin.debug:
        msg: " this task has an always tag"
      tags: 'always'
```

Obwohl der Aufruf mit `--tags firsttags` erfolgt, werden beide Tasks ausgeführt:

```
# ansible-playbook tagbook.yml --tags "firsttag"

PLAY [a playbook to tag around] **

TASK [Gathering Facts] **
ok: [centos1]

TASK [first task with] **
ok: [centos1] => {
    "msg": "I am tagged"
}

TASK [this task has an always tag] **
ok: [centos1] => {
    "msg": " this task has an always tag"
}

PLAY RECAP **
centos1 : ok=3     changed=0      unreachable=0     failed=0
```

Mit dem Tag `never` können Tasks so gekennzeichnet werden, dass sie nie ausgeführt werden, außer wenn sie über ein weiteres Tag explizit aufgerufen werden:

```
tasks:
  - name: this task has a never tag
    ansible.builtin.debug:
      msg: " this task has a never tag"
    tags: [ 'never', 'run' ]
```

Dieser Task wird nur dann ausgeführt, wenn er über mindestens ein weiteres Tag verfügt, das explizit aufgerufen wird. Bei Aufruf von `run` ohne die Option `--tags` wird der Task nicht ausgeführt.

Durch die Argumente `tagged`, `untagged` oder `all` der Option `--tags` können Sie das Verhalten aller Tasks beeinflussen. Mit `tagged` führen Sie alle Tasks aus, die über ein Tag verfügen. Mit `untagged` führen Sie alle Tasks aus, die nicht mit Tags versehen sind. Mit `all` führen Sie alle Tasks aus, die in dem Playbook enthalten sind, außer jenen, die mit dem Tag `never` versehen sind. Ohne Angabe der Option `--tags` werden Playbooks standardmäßig so ausgeführt, als wäre „`--tags all`“ gesetzt.

# Spezielle Optionen für Tags

Spezielle Optionen für Tags:

- --list-tags
- --skip-tags

## Playbookaufruf mit --list-tags

```
# ansible-playbook tagbook.yml --list-tags

playbook: tagbook.yml

play #1 (centos1): a playbook to tag around TAGS: []
    TASK TAGS: [anothertag, firsttag, secondtag]
```

## Spezielle Optionen für Tags

Für den Aufruf von `ansible-playbook` gibt es zwei spezielle Optionen für Tags. Mit der ersten Option `--list-tags` lassen Sie sich anzeigen, welche Tags in einem Playbook vergeben wurden:

```
# ansible-playbook tagbook.yml --list-tags

playbook: tagbook.yml

play #1 (centos1): a playbook to tag around TAGS: []
    TASK TAGS: [anothertag, firsttag, secondtag]
```

Es ist zu beachten, dass mit `--list-tags` nur die von Ihnen vergebenen Tags, und nicht die von Ansible vordefinierten Tags ausgegeben werden.

Mit der Option `--skip-tags` können bestimmte Tags angegeben werden, welche beim Ausführen des Playbooks übersprungen werden sollen:

```
# ansible-playbook tagbook.yml --skip-tags "always,firsttag"
```

Mit der Option `--skip-tags` können auch Tasks übersprungen werden, die mit dem Tag `always` gekennzeichnet wurden.

### Kombination von --tags, --skip-tags und --list-tasks

Sie können die Optionen `--tags` und `--skip-tags` auch mit der Option `--list-tasks` kombinieren. In Kombination werden Ihnen dann nur die Tasks angezeigt, welche nach dem Überspringen der Tags übrig bleiben. So können Sie sich vergewissern, welche Tasks schließlich ausgeführt werden.

Wenn Sie `ansible-playbook` mit der Option `--list-tasks` ausführen, werden Ihnen alle Tasks samt deren Tags – ausgenommen solche mit dem Tag `never` – angezeigt:

```
# ansible-playbook tagbook.yml --list-tasks

playbook: tagbook.yml

play #1 (centos1): a playbook to tag around    TAGS: []
  tasks:
    first task with    TAGS: [anothertag, firsttag]
    a task with a tag TAGS: [secondtag]
    yet another task  TAGS: [anothertag]
```

Wenn Sie dem Befehl nun die Option `--tags` mit dem Tag `anothertag` hinzufügen, so werden nur noch die Tasks angezeigt, die mit diesem Tag gekennzeichnet sind:

Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de

```
# ansible-playbook tagbook.yml --list-tasks --tags "anothertag"

playbook: tagbook.yml

play #1 (centos1): a playbook to tag around    TAGS: []
  tasks:
    first task with    TAGS: [anothertag, firsttag]
    yet another task  TAGS: [anothertag]
```

Wenn Sie die Option `--list-tasks` mit der Option `--skip-tags` und dem Tag `anothertag` verwenden, so werden alle Tasks, außer denen mit `anothertag` gekennzeichneten, aufgelistet:

```
# ansible-playbook tagbook.yml --list-tasks --skip-tags "anothertag"

playbook: tagbook.yml

play #1 (centos1): a playbook to tag around    TAGS: []
  tasks:
    a task with a tag TAGS: [secondtag]
```

## 17 Ansible Advanced

Sie können auf diese Art auch mehrere Tags angeben, sowie auch Tasks mit dem Tag never auflisten:

```
# ansible-playbook tagbook.yml --list-tasks --tags "never,firstrtag"

playbook: tagbook.yml

play #1 (centos1): a playbook to tag around    TAGS: []
  tasks:
    first task with    TAGS: [anotherTag, firstrtag]
    this task has a never tag TAGS: [never, run]
```

### 17.13.1 Aufgabenteil



## Aufgabenteil – Tags

- ① Schreiben Sie ein Playbook mit drei Tasks für alle Hosts. Versehen Sie jeden Task mit einem Tag, der dem jeweiligen Hostnamen entspricht.
- ② Führen Sie das Playbook für die Tags `ubuntu1` und `centos1` aus.
- ③ Fügen Sie dem Playbook einen vierten Task hinzu, der den Ansible Fact `ansible_hostname` für alle Systeme ausgibt. Kennzeichnen Sie diesen Task mit den Tags `never` und `allofthem`. Führen Sie das Playbook so aus, dass nur dieser Task angewendet wird.
- ④ Lassen Sie sich die Tasks für die Tags `ubuntu1` und `never` anzeigen.

1. Schreiben Sie ein Playbook mit drei Tasks für alle Hosts, die jeweils mit dem `ansible.builtin.debug`-Modul den Ansible Fact `ansible_hostname` ausgeben sollen. Erstellen Sie für die Tasks jeweils eine Bedingung, wonach der erste Task nur für den Host `ubuntu1`, der zweite nur für den Host `suse1` und der dritte Task nur für den Host `centos1` ausgeführt wird. Versehen Sie jeden Task mit einem Tag, der dem jeweiligen Hostnamen entspricht.
2. Führen Sie das Playbook für die Tags `ubuntu1` und `centos1` aus.
3. Fügen Sie dem Playbook einen vierten Task hinzu, der den Ansible Fact `ansible_hostname` für alle Systeme ausgibt. Kennzeichnen Sie diesen Task mit den Tags `never` und `allofthem`. Führen Sie das Playbook so aus, dass nur dieser Task angewendet wird.
4. Lassen Sie sich die Tasks für die Tags `ubuntu1` und `never` anzeigen.

## Musterlösung

- Schreiben Sie ein Playbook mit drei Tasks für alle Hosts, die jeweils mit dem `ansible.builtin.debug`-Modul den Ansible Fact `ansible_hostname` ausgeben sollen. Erstellen Sie für die Tasks jeweils eine Bedingung, wonach der erste Task nur für den Host `ubuntu1`, der zweite nur für den Host `suse1` und der dritte Task nur für den Host `centos1` ausgeführt wird. Versehen Sie jeden Task mit einem Tag, der dem jeweiligen Hostname entspricht:

```
---
- name: a playbook to be tagged
  hosts: all
  tasks:

    - name: the first task for ubuntu1
      ansible.builtin.debug:
        msg: "{{ ansible_hostname }}"
      when: ansible_hostname == "ubuntu1"
      tags: ubuntu1

    - name: the second task for suse1
      ansible.builtin.debug:
        msg: "{{ ansible_hostname }}"
      when: ansible_hostname == "suse1"
      tags: suse1

    - name: the third task for centos1
      ansible.builtin.debug:
        msg: "{{ ansible_hostname }}"
      when: ansible_hostname == "centos1"
      tags: centos1
```

- Führen Sie das Playbook für die Tags `ubuntu1` und `centos1` aus:

Rufen Sie hierfür die Tags mit der Option `--tags` auf:

```
# ansible-playbook first_exercise.yml --tags "ubuntu1,centos1"

PLAY [a playbook to be tagged] **

TASK [Gathering Facts] **
ok: [ubuntu1]
ok: [ubuntu2]
ok: [suse2]
ok: [suse1]
ok: [centos1]
ok: [centos2]

TASK [the first task for ubuntu1] **
skipping: [centos1]
skipping: [centos2]
skipping: [suse1]
skipping: [suse2]
skipping: [ubuntu2]
```

```

ok: [ubuntul] => {
    "msg": "ubuntul"
}

TASK [the third taks for centos1] **
skipping: [centos2]
skipping: [suse1]
skipping: [suse2]
skipping: [ubuntul]
ok: [centos1] => {
    "msg": "centos1"
}
skipping: [ubuntu2]

PLAY RECAP **
centos1      : ok=2    changed=0    unreachable=0    failed=0
centos2      : ok=1    changed=0    unreachable=0    failed=0
suse1        : ok=1    changed=0    unreachable=0    failed=0
suse2        : ok=1    changed=0    unreachable=0    failed=0
ubuntul      : ok=2    changed=0    unreachable=0    failed=0
ubuntu2      : ok=1    changed=0    unreachable=0    failed=0

```

3. Fügen Sie dem Playbook einen vierten Task hinzu, der den Ansible Fact `ansible_hostname` für alle Systeme ausgibt. Kennzeichnen Sie diesen Task mit den Tags `never` und `allofthem`. Führen Sie das Playbook so aus, dass nur dieser Task angewendet wird:

Erstellen Sie zunächst den Task:

```

---
- name: a playbook to be tagged
  hosts: all
  tasks:

    - name: the first taks for ubuntul
      ansible.builtin.debug:
        msg: "{{ ansible_hostname }}"
      when: ansible_hostname == "ubuntul"
      tags: ubuntul

    - name: the second taks for suse1
      ansible.builtin.debug:
        msg: "{{ ansible_hostname }}"
      when: ansible_hostname == "suse1"
      tags: suse1

    - name: the third taks for centos1
      ansible.builtin.debug:
        msg: "{{ ansible_hostname }}"
      when: ansible_hostname == "centos1"
      tags: centos1

    - name: the task that is never run
      ansible.builtin.debug:

```

```
msg: "{{ ansible_hostname }}"
tags: [ 'never', 'allofthem' ]
```

Führen Sie anschließend das Playbook mit der Option --tags und dem Argument allofthem aus:

```
# ansible-playbook first_exercise.yml --tags "allofthem"
```

```
PLAY [a playbook to be tagged] **
```

```
TASK [Gathering Facts] **
ok: [ubuntu1]
ok: [ubuntu2]
ok: [suse1]
ok: [suse2]
ok: [centos2]
ok: [centos1]
```

```
TASK [the task that is never run] **
ok: [centos1] => {
    "msg": "centos1"
}
ok: [centos2] => {
    "msg": "centos2"
}
ok: [ubuntu1] => {
    "msg": "ubuntu1"
}
ok: [ubuntu2] => {
    "msg": "ubuntu2"
}
ok: [suse1] => {
    "msg": "suse1"
}
ok: [suse2] => {
    "msg": "suse2"
}
```

```
PLAY RECAP **
```

centos1	:	ok=2	changed=0	unreachable=0	failed=0
centos2	:	ok=2	changed=0	unreachable=0	failed=0
suse1	:	ok=2	changed=0	unreachable=0	failed=0
suse2	:	ok=2	changed=0	unreachable=0	failed=0
ubuntu1	:	ok=2	changed=0	unreachable=0	failed=0
ubuntu2	:	ok=2	changed=0	unreachable=0	failed=0

**4. Lassen Sie sich die Tasks für die Tags `ubuntu1` und `never` anzeigen:**

Rufen Sie das Playbook mit der Option `--list-tasks` und der Option `--tags` mit `ubuntu1` und `never` als Argumenten auf:

```
# ansible-playbook first_exercise.yml --list-tasks --tags
  ↪ "ubuntu1,never"

playbook: first_exercise.yml

play #1 (all): a playbook to be tagged          TAGS: []
  tasks:
    the first task for ubuntu1      TAGS: [ubuntu1]
    the task that is never run      TAGS: [allofthem, never]
```

## 17.14 Strategies



# Strategies

Bestimmen wie ein Playbook abgearbeitet wird mit:

- `strategy:`
- `serial:`

## Auszug aus einem Playbook:

```
---
- name: A play with controlled action
  hosts: all
  strategy: free
  serial: 2
  tasks:
```

Mit `strategy` und `serial` stellt Ansible zwei Werkzeuge zur Ablaufsteuerung der Ausführung von Plays zu Verfügung.

### 17.14.1 Strategy



## Strategy

- mit `strategy` können Sie festlegen, wie ein Play abgearbeitet werden soll
- Plugins:
  - `linear`
  - `free`
  - `debug`
  - `host_pinned`

### Auszug aus einem Play:

```
- name: another strategy
  hosts: all
  strategy: host_pinned
  tasks:
```

Sie können mit `strategy` verschiedene Plugins laden, um den Ablauf von Plays zu steuern. In Form dieser Plugins stehen Ihnen vier verschiedene Strategien zu Verfügung:

PLUGIN:	EFFEKT:
<code>debug</code>	Führt Tasks in einer interaktiven Debugsitzung aus.
<code>free</code>	Führt Plays (und Tasks) unabhängig auf allen Systemen aus.
<code>host_pinned</code>	Arbeitet jeweils nacheinander alle Tasks für einen Host ab.
<code>linear</code>	Führt alle Tasks auf allen Systemen linear nacheinander aus (Vorgabe).

### Das Plugin `linear`

Standardmäßig werden Plays in Playbooks `linear` abgearbeitet. Dies bedeutet, dass Ansible einen Task auf allen im Play referenzierten Zielsystemen parallel ausführt und anschließend mit den folgenden Tasks in der selben Weise verfährt.

Der Eintrag im Playbook:

```
- name: another strategy
  hosts: all
```

## 17 Ansible Advanced

```
strategy: linear
tasks:
```

Das Resultat:

```
PLAY [another strategy] **

TASK [Gathering Facts] **
ok: [ubuntu1]
ok: [ubuntu2]
ok: [suse1]
ok: [suse2]
ok: [centos2]
ok: [centos1]

TASK [command] **
changed: [ubuntu1]
changed: [ubuntu2]
changed: [centos2]
changed: [centos1]
changed: [suse1]
changed: [suse2]

TASK [debug] **
ok: [centos1] => {
    "msg": "uid=0(root) gid=0(root) groups=0(root)"
}
ok: [centos2] => {
    "msg": "uid=0(root) gid=0(root) groups=0(root)"
}
ok: [ubuntu1] => {
    "msg": "uid=0(root) gid=0(root) groups=0(root)"
}
ok: [ubuntu2] => {
    "msg": "uid=0(root) gid=0(root) groups=0(root)"
}
ok: [suse1] => {
    "msg": "uid=0(root) gid=0(root) groups=0(root)"
}
ok: [suse2] => {
    "msg": "uid=0(root) gid=0(root) groups=0(root)"
}

PLAY RECAP **
centos1      : ok=3    changed=1    unreachable=0    failed=0
centos2      : ok=3    changed=1    unreachable=0    failed=0
suse1        : ok=3    changed=1    unreachable=0    failed=0
suse2        : ok=3    changed=1    unreachable=0    failed=0
ubuntu1     : ok=3    changed=1    unreachable=0    failed=0
ubuntu2     : ok=3    changed=1    unreachable=0    failed=0
```

## Das Plugin free

Mit dem Strategie-Plugin `free` werden die Tasks auf allen Systemen gleichzeitig und unabhängig voneinander ausgeführt. Dies kann den Vorteil bieten, dass nicht auf langsame oder fehlerhafte Zielsysteme gewartet werden muss.

Der Eintrag im Playbook:

```
- name: another strategy
  hosts: all
  strategy: free
  tasks:
```

Das Resultat:

```
PLAY [another strategy] **

TASK [Gathering Facts] **
ok: [ubuntu1]
ok: [ubuntu2]

TASK [command] **
changed: [ubuntu1]
changed: [ubuntu2]

TASK [debug] **
ok: [ubuntu1] => {
    "msg": "uid=0(root) gid=0(root) groups=0(root)"
}
ok: [ubuntu2] => {
    "msg": "uid=0(root) gid=0(root) groups=0(root)"
}

TASK [Gathering Facts] **
ok: [suse1]
ok: [suse2]
ok: [centos2]
ok: [centos1]

TASK [command] **
changed: [suse1]

TASK [debug] **
ok: [suse1] => {
    "msg": "uid=0(root) gid=0(root) groups=0(root)"
}

TASK [command] **
changed: [suse2]

TASK [debug] **
ok: [suse2] => {
    "msg": "uid=0(root) gid=0(root) groups=0(root)"
}
```

```
TASK [command] **
changed: [centos2]
changed: [centos1]

TASK [debug] **
ok: [centos2] => {
    "msg": "uid=0(root) gid=0(root) groups=0(root)"
}
ok: [centos1] => {
    "msg": "uid=0(root) gid=0(root) groups=0(root)"
}

PLAY RECAP ***
centos1      : ok=3    changed=1    unreachable=0    failed=0
centos2      : ok=3    changed=1    unreachable=0    failed=0
suse1        : ok=3    changed=1    unreachable=0    failed=0
suse2        : ok=3    changed=1    unreachable=0    failed=0
ubuntu1      : ok=3    changed=1    unreachable=0    failed=0
ubuntu2      : ok=3    changed=1    unreachable=0    failed=0
```

Die Tasks werden alle „durcheinander“ abgearbeitet.

### Das Plugin `host_pinned`

Das Plugin `host_pinned` sorgt dafür, dass die Tasks pro Zielsystem nacheinander abgearbeitet werden.

Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de

Der Eintrag im Playbook:

```
- name: another strategy
  hosts: all
  strategy: host_pinned
  tasks:
```

Das Resultat:

```
PLAY [another strategy] **

TASK [Gathering Facts] **
ok: [ubuntu1]

TASK [Gathering Facts] **
ok: [ubuntu2]
```

```

TASK [command] **
changed: [ubuntul]

TASK [debug] **
ok: [ubuntul] => {
    "msg": "uid=0(root) gid=0(root) groups=0(root)"
}

TASK [command] **
changed: [ubuntu2]

TASK [debug] **
ok: [ubuntu2] => {
    "msg": "uid=0(root) gid=0(root) groups=0(root)"
}

TASK [debug] **
ok: [suse1]

TASK [command] ***

TASK [command] **
ok: [suse2]

TASK [command] **

TASK [command] ***
ok: [centos1]
ok: [centos2]

TASK [command] **

TASK [command] **
changed: [suse1]

TASK [debug] **
ok: [suse1] => {
    "msg": "uid=0(root) gid=0(root) groups=0(root)"
}

TASK [debug] **
changed: [centos1]

TASK [debug] **
ok: [centos1] => {
    "msg": "uid=0(root) gid=0(root) groups=0(root)"
}

TASK [debug] **
ok: [centos2] => {
    "msg": "uid=0(root) gid=0(root) groups=0(root)"
}

```

```
}
```

```
TASK [debug] **
changed: [suse2]
```

```
TASK [debug] **
ok: [suse2] => {
    "msg": "uid=0(root) gid=0(root) groups=0(root)"
}
```

```
PLAY RECAP ***
centos1      : ok=3    changed=1    unreachable=0    failed=0
centos2      : ok=3    changed=1    unreachable=0    failed=0
suse1        : ok=3    changed=1    unreachable=0    failed=0
suse2        : ok=3    changed=1    unreachable=0    failed=0
ubuntul     : ok=3    changed=1    unreachable=0    failed=0
ubuntu2     : ok=3    changed=1    unreachable=0    failed=0
```

### Das Plugin debug

Mit dem Strategie-Plugin `debug` werden die Tasks in einer interaktiven Sitzung ausgeführt. Im Falle eines Fehlers wird Ihnen eine Konsole zu Verfügung gestellt, mit der Sie den Fehler in dem Task korrigieren können.

Der Eintrag im Playbook:

```
- name: another strategy
  hosts: centos2
  strategy: debug
  tasks:
```

Im Folgenden soll mit dem Modul `file` eine Datei im Verzeichnis `/root/` auf dem Ziel-  
system `centos2` angelegt werden:

```
- name: another strategy
  hosts: centos2
  strategy: debug
  tasks:
    - ansible.builtin.file:
        state: touch
        dest: /root/wrong_folder/12idf24545.txt
```

Allerdings ist im Parameter `dest` ein nicht existierender Pfad eingetragen, sodass das Play normalerweise an dieser Stelle unterbrochen würde. Durch die Strategie `debug` bietet Ansible an dieser Stelle nun jedoch eine Debugkonsole:

```
# ansible-playbook strategie_playbook.yml
```

```
PLAY [another strategy] **

TASK [Gathering Facts] **
ok: [centos2]
```

```

TASK [file] **
fatal: [centos2]: FAILED! => {"changed": false, "msg": "Error, could
    ↪ not touch target: [Errno 2] No such file or directory:
    ↪ '/root/wrong_folder/12idf24545.txt'", "path":
    ↪ "/root/wrong_folder/12idf24545.txt", "state": "absent"}
[centos2] TASK: file (debug) >

```

Mit dem Befehl `task.args` können Sie sich die Details des fehlerhaften Tasks ausgeben lassen:

```

[centos2] TASK: file (debug) > task.args
{'_ansible_version': '2.7.9', '_ansible_selinux_special_fs': ['fuse',
    ↪ 'nfs', 'vboxsf', 'ramfs', '9p'], '_ansible_no_log': False,
    ↪ u'dest': u'/root/wrong_folder/12idf24545.txt',
    ↪ '_ansible_module_name': u'file', '_ansible_debug': False,
    ↪ '_ansible_verbosity': 0, '_ansible_keep_remote_files': False,
    ↪ '_ansible_syslog_facility': u'LOG_USER', '_ansible_socket':
    ↪ None, u'state': u'touch', '_ansible_diff': False,
    ↪ '_ansible_remote_tmp': u'~/ ansible/tmp',
    ↪ '_ansible_shell_executable': u'/bin/sh', '_ansible_check_mode':
    ↪ False, '_ansible_tmpdir':
    ↪ u'/root/.ansible/tmp/ansible-tmp-1569488710.73-184006310616165/'}

```

Anschließend können Sie die fehlerhaften Argumente mit dem Befehl `task.args` und der Angabe des korrekten Arguments überschreiben:

```
[centos2] TASK: file (debug) > task.args['dest']='/root/12idf24545.txt'
```

Mit dem Befehl `redo` fordern Sie Ansible dazu auf, den Task erneut auszuführen:

```

[centos2] TASK: file (debug) > redo
changed: [centos2]

PLAY RECAP ****
centos2      : ok=1      changed=1      unreachable=0      failed=0

```

...woraufhin dieser erfolgreich ausgeführt wird.



### Hinweis Anzeigen von Strategie-Plugins

Mit dem Befehl `ansible-doc -t strategy -l` können Sie sich eine Liste der verfügbaren Strategieplugins anzeigen lassen.

Mit `ansible-doc -t strategy <Name des Plugins>` erhalten Sie Details zum angegebenen Plugin.

### 17.14.2 Serial



## Serial

- **serial:** Zusammenfassung von Zielsystemen zu Einheiten, in denen die Tasks abgearbeitet werden
- Werte:
  - absolut
  - in Prozent
  - als Listen

### Auszug aus einem Play:

```
- name: this is serial!
hosts: all
serial:
  - 2
  - '60%'
tasks:
```

Normalerweise arbeitet Ansible die Tasks auf allen im Play referenzierten Systemen parallel ab. Mit dem Schlüsselwort `serial` können Sie festlegen, auf wie vielen Systemen die Tasks zur selben Zeit ausgeführt werden sollen. Hierfür wird die Anzahl zu Beginn eines Plays mit `serial:` festgelegt.

### Serial mit absoluten Werten

Wenn Sie einen Wert von `serial: 2` angeben, so wird das Play immer für Einheiten von zwei Zielsystemen zur selben Zeit ausgeführt:

```
---
- name: are you serial?
hosts: all
serial: 2
tasks:
  - ansible.builtin.debug:
    msg: "This is {{ ansible_host }}."
```

**Das Resultat:**

```
# ansible-playbook strategie_playbook.yml

PLAY [are you serial?] **

TASK [Gathering Facts] **
ok: [centos2]
ok: [centos1]

TASK [debug] **
ok: [centos1] => {
    "msg": "This is centos1."
}
ok: [centos2] => {
    "msg": "This is centos2."
}

PLAY [are you serial?] **

TASK [Gathering Facts] **
ok: [ubuntul]
ok: [ubuntu2]

TASK [debug] **
ok: [ubuntul] => {
    "msg": "This is ubuntul."
}
ok: [ubuntu2] => {
    "msg": "This is ubuntu2."
}

PLAY [are you serial?] **

TASK [Gathering Facts] **
ok: [suse2]
ok: [suse1]

TASK [debug] **
ok: [suse1] => {
    "msg": "This is suse1."
}
ok: [suse2] => {
    "msg": "This is suse2."
}

PLAY RECAP **
centos1      : ok=2    changed=0    unreachable=0    failed=0
centos2      : ok=2    changed=0    unreachable=0    failed=0
suse1        : ok=2    changed=0    unreachable=0    failed=0
suse2        : ok=2    changed=0    unreachable=0    failed=0
ubuntul     : ok=2    changed=0    unreachable=0    failed=0
ubuntu2     : ok=2    changed=0    unreachable=0    failed=0
```

### Serial mit Listen von absoluten Werten

Sie können unter `serial` auch eine Liste mit verschiedenen Werten angeben, wobei der letzte Wert auf die verbleibenden Tasks angewendet wird:

```
---
- name: are you serial?
  hosts: all
  serial:
    - 3
    - 1
  tasks:
    - ansible.builtin.debug:
        msg: "This is {{ ansible_host }}."
```

Das ausgeführte Playbook:

```
# ansible-playbook strategie_playbook.yml

PLAY [are you serial?] **

TASK [Gathering Facts] **
ok: [ubuntul]
ok: [centos1]
ok: [centos2]

TASK [debug] **
ok: [centos1] => {
    "msg": "This is centos1."
}
ok: [centos2] => {
    "msg": "This is centos2."
}
ok: [ubuntul] => {
    "msg": "This is ubuntul."
}

PLAY [are you serial?] **

TASK [Gathering Facts] **
ok: [ubuntu2]

TASK [debug] **
ok: [ubuntu2] => {
    "msg": "This is ubuntu2."
}

PLAY [are you serial?] **

TASK [Gathering Facts] **
ok: [suse1]

TASK [debug] **
ok: [suse1] => {
```

```

        "msg": "This is suse1."
}

PLAY [are you serial?] **

TASK [Gathering Facts] **
ok: [suse2]

TASK [debug] **
ok: [suse2] => {
    "msg": "This is suse2."
}

PLAY RECAP **
centos1      : ok=2    changed=0    unreachable=0    failed=0
centos2      : ok=2    changed=0    unreachable=0    failed=0
suse1        : ok=2    changed=0    unreachable=0    failed=0
suse2        : ok=2    changed=0    unreachable=0    failed=0
ubuntu1      : ok=2    changed=0    unreachable=0    failed=0
ubuntu2      : ok=2    changed=0    unreachable=0    failed=0

```

Hier wird das Play zunächst auf drei Zielsystemen parallel und anschließend jeweils auf einem Zielsystem ausgeführt.

### Serial mit Prozentangaben

Alternativ zu den absoluten Werten können die Werte für `serial` auch in Prozent angegeben werden:

```

---
- name: are you serial?
  hosts: all
  serial:
    - '30%'
    - '50%'
  tasks:
    - debug:
        msg: "This is {{ ansible_host }}."

```

Mit diesen Angaben wird das Play zunächst für 30% und anschließend für jeweils 50% der verfügbaren Hosts ausgeführt.

```

# ansible-playbook strategie_playbook.yml

PLAY [are you serial?] **

TASK [Gathering Facts] **
ok: [centos1]

TASK [debug] **
ok: [centos1] => {
    "msg": "This is centos1."
}

```

```
PLAY [are you serial?] **

TASK [Gathering Facts] **
ok: [ubuntu1]
ok: [ubuntu2]
ok: [centos2]

TASK [debug] **
ok: [centos2] => {
    "msg": "This is centos2."
}
ok: [ubuntu1] => {
    "msg": "This is ubuntu1."
}
ok: [ubuntu2] => {
    "msg": "This is ubuntu2."
}

PLAY [are you serial?] **

TASK [Gathering Facts] **
ok: [suse2]
ok: [suse1]

TASK [debug] **
ok: [suse1] => {
    "msg": "This is suse1."
}
ok: [suse2] => {
    "msg": "This is suse2."
}

PLAY RECAP **
centos1      : ok=2    changed=0    unreachable=0    failed=0
centos2      : ok=2    changed=0    unreachable=0    failed=0
suse1        : ok=2    changed=0    unreachable=0    failed=0
suse2        : ok=2    changed=0    unreachable=0    failed=0
ubuntu1     : ok=2    changed=0    unreachable=0    failed=0
ubuntu2     : ok=2    changed=0    unreachable=0    failed=0
```

### Serial mit gemischten Listen aus absoluten und relativen Werten

Sie können die absoluten und relativen Werte auch kombinieren:

```
---
- name: are you serial?
  hosts: all
  serial:
    - 1
    - '40%'
  tasks:
    - ansible.builtin.debug:
```

```
msg: "This is {{ ansible_host }}."
```

### Ausführung des Playbooks:

```
PLAY [are you serial?] **

TASK [Gathering Facts] **
ok: [centos1]

TASK [debug] **
ok: [centos1] => {
    "msg": "This is centos1."
}

PLAY [are you serial?] **

TASK [Gathering Facts] **
ok: [ubuntul]
ok: [centos2]

TASK [debug] **
ok: [centos2] => {
    "msg": "This is centos2."
}
ok: [ubuntul] => {
    "msg": "This is ubuntul."
}

PLAY [are you serial?] **

TASK [Gathering Facts] **
ok: [ubuntu2]
ok: [suse1]

TASK [debug] **
ok: [ubuntu2] => {
    "msg": "This is ubuntu2."
}
ok: [suse1] => {
    "msg": "This is suse1."
}

PLAY [are you serial?] **

TASK [Gathering Facts] **
ok: [suse2]

TASK [debug] **
ok: [suse2] => {
    "msg": "This is suse2."
}

PLAY RECAP **
centos1 : ok=2      changed=0      unreachable=0      failed=0
```

```
centos2      : ok=2    changed=0    unreachable=0    failed=0
suse1        : ok=2    changed=0    unreachable=0    failed=0
suse2        : ok=2    changed=0    unreachable=0    failed=0
ubuntul     : ok=2    changed=0    unreachable=0    failed=0
ubuntu2     : ok=2    changed=0    unreachable=0    failed=0
```

### Fehlerquote für Einheiten

Sind in mit `serial` zusammengefassten Einheiten Tasks enthalten, die bei Ausführung einen Fehler melden, so bricht Ansible das Ausführen nicht ab, solange die Tasks auf mindestens einem Host fehlerfrei ausgeführt werden. Mit der Option `max_fail_percentage` können Sie angeben, ab wie viel Prozent Fehlerquote der Vorgang abgebrochen werden soll:

```
---
- name: are you serial?
  hosts: all
  serial: 15
  max_fail_percentage: 30
  tasks:
    - ansible.builtin.debug:
        msg: "This is {{ ansible_host }}."
```

Wenn das Ausführen der Tasks bei mehr als 30% der sich in der Einheit befindlichen Zielsysteme fehlschlägt, wird das Ausführen des Playbooks abgebrochen.



#### Hinweis Die Option `forks`

Mit der Option `--forks` bzw. `-f` können Sie bestimmen, wie viele Zielsysteme von Ansible parallel maximal angesprochen werden. Standardmäßig ist hier die Anzahl von fünf Zielsystemen festgelegt. Sie können diesen Wert entweder dauerhaft in der `ansible.cfg` unter dem Eintrag `forks` oder für den jeweiligen Ansible-Befehl mit der Option `--forks` bzw. `-f` ändern. Der Wert von `forks` limitiert ggf. die Möglichkeiten von `serial`.

### 17.14.3 Aufgabenteil



## Aufgabenteil – Serial

- ① Schreiben Sie ein Playbook für alle Hosts, in dem ein erster Task mit dem `ansible.builtin.command`-Modul ausliest, welche Dateien sich im Verzeichnis `/etc/ansible` befinden, und das Ergebnis in einer Variable registriert. Schreiben Sie einen zweiten Task, der mit dem `ansible.builtin.debug`-Modul den Verzeichnisinhalt der in der registrierten Variable gespeicherten Daten ausgibt. Führen Sie das Playbook so aus, dass alle Hosts gleichzeitig angesprochen werden und diese nicht gegenseitig aufeinander warten.
- ② Ändern Sie das Playbook so ab, dass die Tasks immer für drei Hosts gleichzeitig ausgeführt werden.

1. Schreiben Sie ein Playbook für alle Hosts, in dem ein erster Task mit dem `ansible.builtin.command`-Modul ausliest, welche Dateien sich im Verzeichnis `/etc/ansible` befinden, und das Ergebnis in einer Variable registriert. Schreiben Sie einen zweiten Task, der mit dem `ansible.builtin.debug`-Modul den Verzeichnisinhalt der in der registrierten Variable gespeicherten Daten ausgibt. Führen Sie das Playbook so aus, dass alle Hosts gleichzeitig angesprochen werden und diese nicht gegenseitig aufeinander warten.
2. Ändern Sie das Playbook so ab, dass die Tasks immer für drei Hosts gleichzeitig ausgeführt werden.

## Musterlösung

- Schreiben Sie ein Playbook für alle Hosts, in dem ein erster Task mit dem `ansible.builtin.command`-Modul ausliest, welche Dateien sich im Verzeichnis `/etc/ansible` befinden, und das Ergebnis in einer Variable registriert. Schreiben Sie einen zweiten Task, der mit dem `ansible.builtin.debug`-Modul den Verzeichnisinhalt der in der registrierten Variable gespeicherten Daten ausgibt. Führen Sie das Playbook so aus, dass alle Hosts gleichzeitig angesprochen werden und diese nicht gegenseitig aufeinander warten:

Verfassen Sie zunächst das Playbook mit dem Task mit dem `ansible.builtin.command`-Modul, der registrierten Variable sowie dem Debug-Task. Legen sie mit `strategy` die Strategie `free` fest:

```
---
- name: a playbook to test some strategies
  hosts: all
  strategy: free
  tasks:
    - name: a task to collect an register some data
      ansible.builtin.command: ls /etc/ansible
      register: var_ls
    - name: show me what you've got
      ansible.builtin.debug:
        msg: "{{ var_ls['stdout_lines'] }}"
```

Wenn Sie das Playbook nun ausführen, so werden die Tasks auf den Hosts ohne bestimmte Reihenfolge abgearbeitet:

```
# ansible-playbook free_playbook.yml

PLAY [a playbook to test some strategies] **

TASK [Gathering Facts] **
ok: [ubuntu1]
ok: [ubuntu2]

TASK [a task to collect an register some data] **
changed: [ubuntu1]
changed: [ubuntu2]

TASK [show me what you've got] **
ok: [ubuntu1] => {
    "msg": [
        "ansible.cfg"
    ]
}
ok: [ubuntu2] => {
    "msg": [
        "ansible.cfg"
    ]
}

TASK [Gathering Facts] **
```

```

ok: [suse2]
ok: [suse1]

TASK [a task to collect an register some data] **
changed: [suse2]
changed: [suse1]

TASK [Gathering Facts] **
ok: [centos1]

TASK [show me what you've got] **
ok: [suse2] => {
    "msg": [
        "ansible.cfg"
    ]
}
ok: [suse1] => {
    "msg": [
        "ansible.cfg"
    ]
}

TASK [Gathering Facts] **
ok: [centos2]

TASK [a task to collect an register some data] **
changed: [centos1]

TASK [show me what you've got] **
ok: [centos1] => {
    "msg": [
        "ansible.cfg"
    ]
}

TASK [a task to collect an register some data] **
changed: [centos2]

TASK [show me what you've got] **
ok: [centos2] => {
    "msg": [
        "ansible.cfg"
    ]
}

PLAY RECAP **
centos1      : ok=3    changed=1    unreachable=0    failed=0
centos2      : ok=3    changed=1    unreachable=0    failed=0
suse1        : ok=3    changed=1    unreachable=0    failed=0
suse2        : ok=3    changed=1    unreachable=0    failed=0
ubuntu1     : ok=3    changed=1    unreachable=0    failed=0
ubuntu2     : ok=3    changed=1    unreachable=0    failed=0

```

**2. Ändern Sie das Playbook so ab, dass die Tasks immer für drei Hosts gleichzeitig ausgeführt werden:**

Entfernen Sie die Zeile `strategy: free` und ersetzen Sie diese durch `serial: 3`:

```
---
- name: a playbook to test some strategies
  hosts: all
  serial: 3
  tasks:
    - name: a task to collect an register some data
      ansible.builtin.command: ls /etc/ansible
      register: var_ls
    - name: show me what you've got
      ansible.builtin.debug:
        msg: "{{ var_ls['stdout_lines'] }}"
```

Wenn Sie das Playbook nun ausführen, so werden alle Tasks für jeweils drei Hosts ausgeführt:

```
# ansible-playbook free_playbook.yml

PLAY [a playbook to test some strategies] **

TASK [Gathering Facts] **
ok: [ubuntu1]
ok: [ubuntu2]
ok: [suse1]

TASK [a task to collect an register some data] **
changed: [ubuntu1]
changed: [ubuntu2]
changed: [suse1]

TASK [show me what you've got] **
ok: [ubuntu1] => {
    "msg": [
        "ansible.cfg"
    ]
}
ok: [ubuntu2] => {
    "msg": [
        "ansible.cfg"
    ]
}
ok: [suse1] => {
    "msg": [
        "ansible.cfg"
    ]
}

PLAY [a playbook to test some strategies] **

TASK [Gathering Facts] **
```

```

ok: [suse2]
ok: [centos1]
ok: [centos2]

TASK [a task to collect an register some data] **
changed: [centos1]
changed: [centos2]
changed: [suse2]

TASK [show me what you've got] **
ok: [suse2] => {
    "msg": [
        "ansible.cfg"
    ]
}
ok: [centos1] => {
    "msg": [
        "ansible.cfg"
    ]
}
ok: [centos2] => {
    "msg": [
        "ansible.cfg"
    ]
}

PLAY RECAP **

centos1      : ok=3    changed=1    unreachable=0    failed=0
centos2      : ok=3    changed=1    unreachable=0    failed=0
suse1        : ok=3    changed=1    unreachable=0    failed=0
suse2        : ok=3    changed=1    unreachable=0    failed=0
ubuntu1     : ok=3    changed=1    unreachable=0    failed=0
ubuntu2     : ok=3    changed=1    unreachable=0    failed=0

```

## 17.15 Error Handling – Übersicht



# Error Handling – Übersicht

Möglichkeiten des Umgangs mit Fehlern in Playbooks:

- ignore\_errors
- any\_errors\_fatal
- failed\_when / changed\_when
- Handler
- Blocks
- Dry-Run mit --check
- Dateiänderungen ausgeben lassen mit --diff
- „Verbose“-Level mit -v bis -vvvv
- Option verbosity des ansible.builtin.debug-Moduls

Ansible hält eine Vielzahl an Möglichkeiten bereit, mit beim Ausführen von Playbooks auftretenden Fehlern umzugehen. Die meisten Werkzeuge zum Umgang mit Fehlern sind in dieser Unterlage in den einzelnen Kapiteln verteilt, dieser Abschnitt soll noch einmal der Übersicht dienen.

### Playbook-Optionen zum Umgang mit Fehlern

Mit der Option ignore\_errors können Sie ansible-playbook veranlassen, ein Play trotz eines auftretenden Fehlers weiter ausführen zu lassen:

```
- ansible.builtin.debug:
  msg: "I use a {{ variable }}"
  ignore_errors: yes
```

Wenn in diesem Fall die Variable variable nicht deklariert ist, wird das Playbook trotzdem weiter ausgeführt.

Mit der Option any\_errors\_fatal können Sie für das genaue Gegenteil von ignore\_errors sorgen und das Ausführen des Plays auch dann abbrechen, wenn der

Fehler eigentlich nur dafür gesorgt hätte, dass der Task lediglich für einen Host übersprungen würde:

```
---
- name: you name it
  hosts: suse
  any_errors_fatal: yes
  roles:
    - role: b1
```

Mit dieser Option wird das Play unterbrochen, sobald es zu einem Fehlerfall kommt.

Die Option `max_fail_percentage` lässt Sie einen Wert in Prozent angeben, mit dem Sie eine Toleranzgrenze für die Anzahl fehlerhafter Tasks festlegen, bevor das Play abgebrochen wird:

```
---
- name: you name it
  hosts: suse
  max_fail_percentage: '30%'
  roles:
    - role: b1
```

Wenn mehr als 30% der Tasks fehlgeschlagen, wird das Play abgebrochen.

### **`changed_when` und `failed_when`**

Mit `changed_when` und `failed_when` können Sie definieren, wann ein Task als erfolgreich ausgeführt oder als gescheitert gewertet wird.

`changed_when`:

```
- name: first task
  ansible.builtin.command: "httpd -k start"
  register: httpd_register
  changed_when: "'already running' not in httpd_register['stdout']"
```

`failed_when`:

```
- name: a task that starts a service via the command module
  ansible.builtin.command: "httpd -k start"
  register: httpd_register
  failed_when: "'already running' not in httpd_register['stdout'] and
    ↴ httpd_register['rc'] !=0"
```

Ausführlichere Erläuterungen finden Sie im Kapitel zu `when`.

## Handler

Wenn ein Task fehlschlägt, an dessen Ende mit `notify` ein Handler ausgeführt werden sollte, so können Sie mit der Option `force_handlers` dafür sorgen, dass die verwendeten Handler auf jeden Fall ausgeführt werden.

Auf der Kommandozeile:

```
# ansible-playbook handler_playbook.yml --force-handlers
```

Im Playbook:

```
---
- name: a playbook with a handler
  hosts: ubuntu
  handlers:
    - name: restart sshd
      ansible.builtin.service:
        name: sshd
        state: restarted
  force_handlers: true
  tasks:
```

In der `ansible.cfg`:

```
force_handlers=true
```

## Blocks

Eine weitere Möglichkeit des Umgangs mit Fehlern ist es, die Tasks in Blocks zusammenzufassen und mit den Optionen `rescue:` und `always:` zu arbeiten, oder andere Optionen an die Tasks in dem Block weiterzuvererben.

```
- name: block
  block:
    - ansible.builtin.debug:
        msg: "first message"
    - ansible.builtin.debug:
        msg: " I use a {{ variable }}"
  rescue:
    - ansible.builtin.debug:
        msg: "In case of error this is shown."
  always:
    - name: I am always executed, no matter what!
      ansible.builtin.debug:
        msg: "Something went wrong here, or not, doesn't matter?!"
```

Ausführlichere Erläuterungen finden Sie im Abschnitt zu `block` auf S. 550 ff.

## Dry-Run und Diff

Die Option `--check` bietet bei der Ausführung von `ansible-playbook` die Möglichkeit einen sogenannten „Dry-Run“ durchzuführen, d. h. ein Playbook auszuführen, ohne dass tatsächlich Änderungen am Zielsystem vorgenommen werden. Die Option stößt allerdings dort an ihre Grenzen, wo das Ausführen von Tasks tatsächlich von Änderungen abhängig ist. Wenn beispielsweise in einem ersten Task ein Paket installiert werden soll und in einem zweiten sichergestellt werden, dass der Dienst dieses Pakets gestartet ist, so wird der zweite Task fehlschlagen, da der Dienst nicht vorhanden ist. Die Paketinstallations hätte eine Änderung am Zielsystem bedeutet und wurde somit nicht durchgeführt.

Mit der Option `--diff` für `ansible-playbook` wird Ihnen ein Diff ausgegeben, wenn Ansible eine Datei z. B. mit dem `lineinfile`-Modul verändert. Das Diff wird nur dann ausgegeben, wenn die Datei auch wirklich geändert wurde.

```
# ansible-playbook -i inventory lineinfile.yml --diff

PLAY [diff] **

TASK [Gathering Facts] **
ok: [suse2]

TASK [make sure a file is present] *
--- before
+++ after
@@ -1,6 +1,6 @@
{
-    "atime": 1647535987.6688104,
-    "mtime": 1647535987.6688104,
+    "atime": 1647535987.670455,
+    "mtime": 1647535987.670455,
        "path": "/root/diff.txt",
-    "state": "absent"
+    "state": "touch"
}

changed: [suse2]

TASK [make sure a line is present in file] **
--- before: /root/diff.txt (content)
+++ after: /root/diff.txt (content)
@@ -0,0 +1 @@
+I am the new line.

changed: [suse2]

PLAY RECAP **
suse2 : ok=3      changed=2      unreachable=0
      ↪ failed=0     skipped=0     rescued=0     ignored=0
```

### Setzen des Verbose-Levels

Durch das Setzen der Option `-v` beim Aufruf von `ansible` oder `ansible-playbook` schalten Sie den *Verbose*-Modus ein und lassen Ansible zusätzliche Informationen des Befehlsaufrufs direkt auf der Kommandozeile ausgeben. Insgesamt kennt Ansible vier Verbose-Level, die Sie durch die Anzahl der `v` in der Option beeinflussen, z. B.:

```
# ansible-playbook playbook.yml -vvv
```

Je höher das angegebene Verbose-Level, desto mehr zusätzliche Informationen gibt Ansible aus.

### Debug-Modul über Verbose-Level aktivieren

Das `ansible.builtin.debug`-Modul kennt den Parameter `verbosity`, über den Sie das Ausführen des Debug-Tasks von der Angabe der Verbose-Option auf der Kommandozeile abhängig machen:

```
- name: a debug task
  ansible.builtin.debug:
    msg: "{{ registered_var }}"
    verbosity: 2
```

In diesem Fall wird der Debug-Task nur dann ausgeführt, wenn das Playbook mit der Option `-vv` oder höher aufgerufen wird. Insgesamt kann dem `verbosity`-Parameter ein Wert zwischen 0 und 4 übergeben werden, wobei 0 dem impliziten Standardwert (immer an) und 4 dem Aufruf mit `-vvvv` entspricht.

## 18 Ansible im Team



# Ansible im Team

## 18.1 Zielsetzung



# Zielsetzung

Dieses Kapitel liefert einen Überblick über die Möglichkeiten, in Teams mit Ansible zu arbeiten:

- Git
- AWX
- Ansible Tower

## 18.2 Ansible im Team



# Ansible im Team

Die Arbeit im Team effizient gestalten mit:

- Git:
  - Code-Struktur im Repository
  - Repositories zentral verwalten mit z. B. Gitea
  - Workflows
- AWX:
  - Code direkt aus Git-Repositories importieren
  - Jobs erstellen und automatisiert ausführen
- Ansible Tower:
  - Funktionalität wie bei AWX
  - Support
  - Release-Zyklen
  - nicht Containerisiert

Bei der Arbeit in größeren Kontexten kommt es häufig vor, dass der Ansible-Code von einem großen Team oder mehreren Teams entwickelt, gepflegt und angewendet wird. Damit verschiedene Personen oder Teams an demselben Code arbeiten können, liegt es nahe, diesen zentral zu verwalten. Hierfür eignen sich z. B. Versionierungssysteme wie SVN oder Git in Kombination mit zentralen Plattformen wie GitLab, Gitea oder GitHub. Obwohl Ansible selbst nicht direkt über Features für die Arbeit in Teams verfügt, kann die Projektstruktur von Ansible auf das Arbeiten in unterschiedlichen Teams und für die Verwaltung in Versionierungssystemen optimiert werden.

Mit Werkzeugen wie Ansible Tower oder dessen Upstream-Variante AWX lässt sich der Ansible-Code zudem direkt aus zentralen Strukturen heraus laden und anwenden. So entsteht eine Schnittstelle, die zwischen der Entwicklung und dem Ausführen des Codes nahezu nahtlos geschlossen werden kann. Das folgende Kapitel wird das Augenmerk auf das Versionierungssystem Git in Kombination mit Gitea legen. Es werden Möglichkeiten gezeigt, Ansible-Code für die Arbeit mit Teams optimiert in Git-Repositories abzulegen. An diesen Repositories wird anschließend die Arbeit mit AWX und Ansible Tower gezeigt.

### 18.3 Ansible im Team mit Git



## Ansible im Team mit Git

mögliche Repository-Strukturen:

- ein Repository für den gesamten Ansible-Code:
  - gut bei „kurzen Wegen“
  - Änderungen können schnell umgesetzt werden
  - skaliert nicht
- ein Repository pro Rolle:
  - lohnenswert bei großen Teams und Projekten
  - detaillierte Zugriffs- und Rechteverwaltung
  - Changemanagement
  - Releasemanagement
  - Abhängigkeiten
  - Testing

Je nach Team- und Projektgröße kann sich die Arbeit an mit Ansible gepflegten Umgebungen, wie auch am Ansible-Code selbst, ganz unterschiedlich gestalten. Je komplexer die Umgebung, desto umfangreicher wird auch der Ansible-Code, der notwendig ist, um die Umgebung einzurichten. Wenn dafür ein großes Team an diesem Code arbeitet oder die Landschaft von mehreren Abteilungen und Teams verwaltet wird, so ist es durchaus sinnvoll, andere Strategien anzuwenden, als wenn es sich um eine kleine Landschaft mit einem einzigen kleinen Team handelt. Es kann die Effizienz der Projekte durchaus positiv beeinflussen, die Arbeitsprozesse an diese strukturellen Gegebenheiten anzupassen. Im Folgenden werden einige Möglichkeiten skizziert, mit denen Arbeitsprozesse im Team mit Ansible gestaltet werden können.

### Ansible und Versionierungssysteme

Ein erster Schritt zur Optimierung der Arbeitsabläufe in der Code-Entwicklung im Team besteht in der Verwendung eines Versionierungssystems wie SVN oder Git. Gilt dies für ziemlich jede Entwicklungsarbeit im Team, so gibt es doch im Umgang mit Ansible bestimmte Strategien, die langfristig zu besseren Ergebnissen führen, als andere. In dieser Unterlage wird auf das Versionierungssystem Git eingegangen.

## Ansible und Git

Git bietet den Vorteil, dass Teams und einzelne Teammitglieder in verschiedenen Branches simultan an verschiedenen Dingen arbeiten können. Jedoch stellt sich die Frage, wie der Ansible-Code zunächst aufgeteilt und ferner auf Repositories im Git verteilt werden soll. Ansible-Code kann von sich aus schon sehr fragmentiert und modular organisiert werden. Werden die Variablen an globalen Orten wie den Host- oder den Groupvars gesetzt oder in den Rollen selbst? Und wie wird diese Codes-Struktur dann ins Git übertragen?

### Flache Prozessstrukturen – Ein Repository für alles

Es besteht grundsätzlich die Möglichkeit, einfach den gesamten Ansible-Code, also Playbooks, Rollen, Inventories sowie Host- und Groupvars in einem einzigen Repository abzulegen. Ein solches Vorgehen bietet sich eher bei kleineren Team- und Projektstrukturen mit kurzen Wegen an. Dies setzt jedoch voraus, dass sich die Teammitglieder regelmäßig austauschen können und wissen, woran die jeweils anderen Teammitglieder gerade arbeiten um z. B. Fehler beim Mergen zu vermeiden. Auf diese Art können Änderungen schnell besprochen und mit wenig Aufwand umgesetzt werden. Auch bietet ein solches Vorgehen den Vorteil, dass die Abhängigkeiten zu Komponenten außerhalb des Repositories gering gehalten werden. Der große Nachteil dieser Strategie ist, dass sie nicht sehr gut skaliert und ab einer gewissen Team- und Projektgröße die Arbeitsprozesse unübersichtlich und schwer nachvollziehbar werden. Je größer das Projekt wird, desto mehr lohnt es sich, die Prozessstrukturen selbst zu einem Gegenstand eines Projektes zu machen.

### Detaillierte Prozessstrukturen – Ein Repository für jede Rolle

Wenn Projekte eine Größe erreichen, bei der sie nicht mehr durch direkte Kommunikation unter den Teammitgliedern oder den Teams zu überschauen sind, so ist es ratsam, die Projektstrukturen für diese Größe zu optimieren. Ein Ansatz dafür besteht z. B. darin, nicht mehr ein einziges Git-Repository für den gesamten Ansible-Code zu haben, sondern für jede einzelne Rolle eigene Repositories anzulegen, welche dann von den jeweiligen Teams oder Mitgliedern bearbeitet werden können. Dies führt anfangs zu mehr Aufwand gegenüber der Methode, alles in einem Repository zu verwalten, birgt insgesamt jedoch weniger Potential für Fehler. Im Optimalfall würde dieses Vorgehen begleitet von einer Kontrollinstanz, die sich ausschließlich mit der Verwaltung der Strukturen sowie dem Release- und Changemanagement befasst, sodass dies nicht zu einer zusätzlichen Aufgabe der Entwickler und Mitarbeiter wird. Auch ist zu beachten, dass sich so bestimmte Abhängigkeiten außerhalb der Rollen selbst bilden, die erfasst, dokumentiert und in separaten Strukturen bereitgestellt werden können. Diese Abhängigkeiten können etwa in anderen Repositories zur Verfügung gestellt und in einer `requirements.yml` eingetragen von Ansible mit abgedeckt werden. Eine solche Projektstruktur bedeutet zwar mehr Aufwand, bietet jedoch auch mehr Überblick über einzelne Komponenten und zudem die Möglichkeit, diese unabhängig voneinander zu pflegen.

Ein nicht zu unterschätzender Aspekt der granularen Aufteilung der Ansible-Strukturen in verschiedene Git-Repositories liegt in der Rechteverwaltung. Wird der gesamte Code in einem einzigen Repository verwaltet, so müssen Rechte global vergeben werden, auch wenn einzelne Personen oder Teams nur an bestimmten Teilen arbeiten sollen. Sind die Strukturen allerdings unterteilt, so entsteht die Möglichkeit, die Rechteverwaltung differenziert zu gestalten und bei Bedarf nur die Rechte zu vergeben, die auch wirklich für die Bearbeitung notwendig sind. So kann sichergestellt werden, dass weder unbefugte noch unbeabsichtigte Zugriffe stattfinden. Zudem können die tatsächlich erfolgten Zugriffe besser nachvollzogen werden, wodurch insgesamt ein besserer Überblick über das gesamte Projekt und dessen Entwicklung gewährleistet ist.

Ebenfalls können mit diesem Vorgehen einzelne Komponenten unabhängig voneinander entwickelt und getestet werden, was bedeutet, dass sich einzelne Teams und Teammitglieder weniger „im Weg“ stehen und effizienter interagieren können. Durch eine feingliedrige Projektstruktur werden die langen Wege, die normalerweise in großen Teams entstehen, um einiges verkürzt. Bei dem Einsatz von ausführlichen Projektstrukturen bietet sich zudem der Einsatz von Werkzeugen wie AWX oder Ansible Tower an, mit denen die angelegten Ansible-Strukturen direkt aus der Versionsverwaltung heraus geladen und auf die Zielsysteme ausgespielt werden können, sodass Entwicklung und Produktion fließend ineinander übergehen.

### 18.3.1 Einrichten von Gitea



## Gitea einrichten

① Rufen Sie `http://git:3000/install` in einem Browser auf:  
**SSH Server Domain:** git  
**Gitea Base Url:** `http://git:3000/`

② Legen Sie einen User „tux“ an.  
③ Hinterlegen Sie Ihren öffentlichen SSH-Key vom Controller.  
④ Erstellen Sie ein Repository „ansible\_training“.

---

B1 Systems GmbH      Terraform & Ansible Grundlagen      275 / 279

In der Schulungsumgebung dieser Schulung befindet sich eine vorinstallierte Gitea-Instanz. Diese Instanz dient dazu, eine Schnittstelle zwischen Ihrem lokalen Git und AWX bereitzustellen. Damit Sie den Inhalt ihres lokalen Git in die Gitea-Instanz hochladen können, müssen Sie diese zunächst einrichten und ein Repository anlegen. Wie Sie Ihr Gitea einrichten wird im Folgenden gezeigt.

### Gitea einrichten

Um Ihr Gitea einzurichten, rufen Sie in einem Webbrowser die folgende URL auf:

`http://git:3000/install`

## 18 Ansible im Team

Es öffnet sich nun eine Seite auf der Sie die Basiseinstellungen für Gitea festlegen müssen:

The screenshot shows the 'Initial Configuration' page for Gitea. It includes sections for Database Settings, General Settings, and Optional Settings, with a prominent 'Install Gitea' button at the bottom.

**Database Settings**

- Database Type: SQLite3
- Path: /data/gitea/gitea.db

**General Settings**

- Site Title: Gitea: Git with a cup of tea
- Repository Root Path: /data/git/repositories
- Git LFS Root Path: /data/git/lfs
- Run As Username: git
- SSH Server Domain: git
- SSH Server Port: 22
- Gitea HTTP Listen Port: 3000
- Gitea Base URL: http://git:3000/
- Log Path: /data/gitea/log

**Optional Settings**

- Email Settings
- Server and Third-Party Service Settings
- Administrator Account Settings

**Install Gitea**

Tragen Sie hier als *SSH Server Domain* `git` und als *Gitea Base URL* `http://git:3000/` ein. Klicken Sie anschließend auf *Install Gitea*.

Öffnen Sie nun die URL

`http://git:3000/user/sign_up`

und erstellen Sie ein Konto:

The screenshot shows a registration form titled "Register". It has four input fields: "Username" (tux), "Email Address" (tux@localhost.localdomain), "Password" (\*\*\*\*\*), and "Re-Type Password" (\*\*\*\*\*). Below the fields is a green "Register Account" button. At the bottom of the form, there is a link "Already have an account? Sign in now!".

Tragen Sie unter *Username* den Namen `tux`, unter *Email Address* die Adresse `tux@localhost.localdomain` und als *Password* `b1systems` ein. Klicken Sie auf *Register Account*.

Sie befinden sich nun auf der Übersichtsseite (*Dashboard*) Ihres neuen Gitea-Kontos:

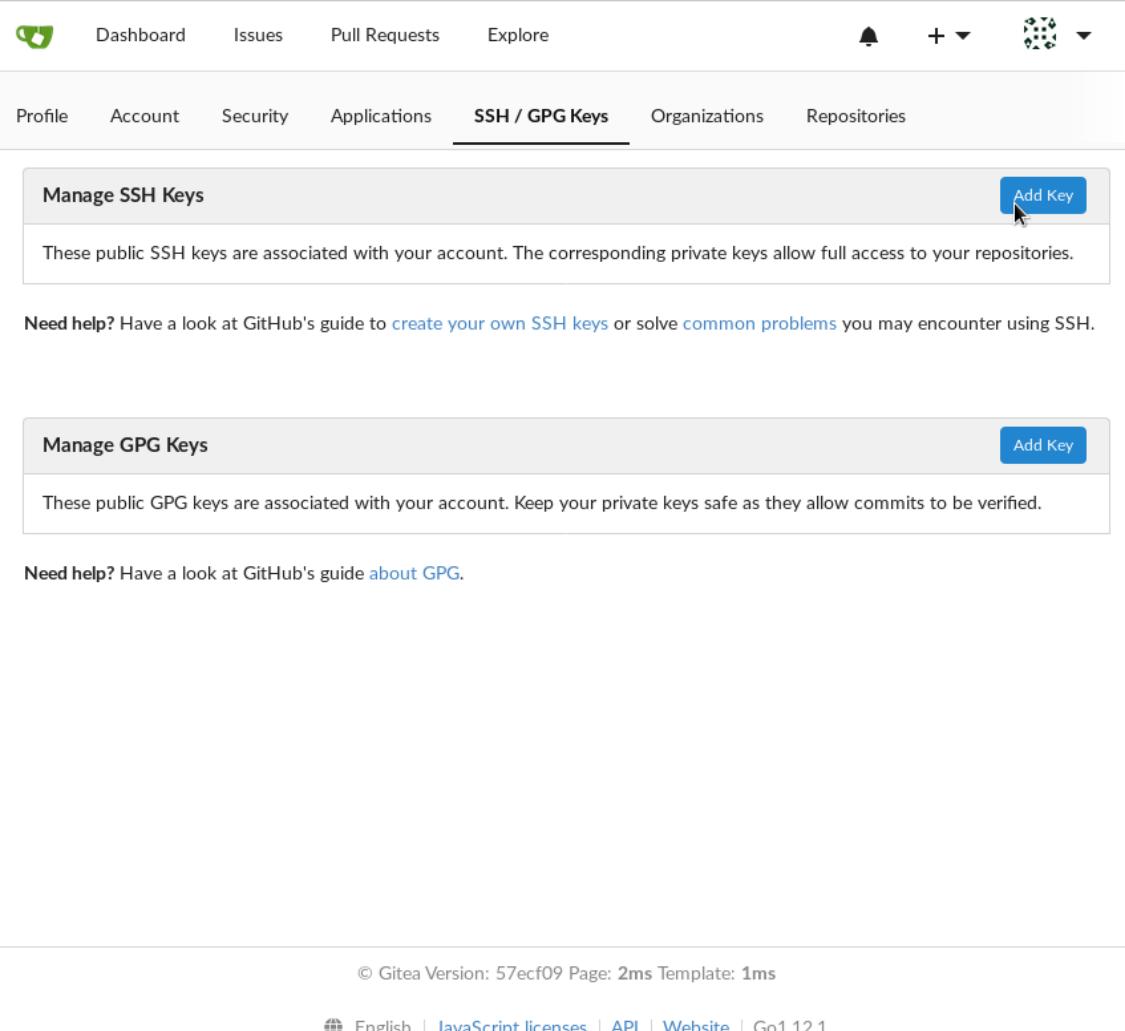
The screenshot shows the Gitea dashboard. At the top, there are navigation links: Dashboard, Issues, Pull Requests, Explore, and a user icon with the name "tux". Below the navigation is a success message: "Account was successfully created.". To the right is a sidebar with tabs for "Repository" and "Organization". Under "Repository", there is a section for "Repositories" (0) with a search bar labeled "Find a repository...". Below the search bar are filters for "All" (0), "Sources", "Forks", "Mirrors", and "Collaborative". At the bottom of the dashboard, there is footer information: "© Gitea Version: 57ecf09 Page: 62ms Template: 57ms", language links ("English | JavaScript licenses | API | Website | Go1.12.1"), and a copyright notice: "© B1 Systems GmbH 2004 – 2024; Course materials may not be reproduced in whole or in part without the written permission of B1 Systems."

## 18 Ansible im Team

Wählen Sie oben rechts im Menü den Punkt *Settings* aus:

The screenshot shows a Gitea interface. At the top, there is a navigation bar with links for Dashboard, Issues, Pull Requests, and Explore. To the right of the navigation bar is a user profile icon labeled 'tux' with a dropdown arrow. On the far right of the header are icons for notifications, a plus sign, and a gear. The main content area shows a repository named 'Repository'. Below it is a section titled 'Repositories' with a search bar and a button for 'Find a repository'. Underneath is a table with two rows: 'All' (with a count of 0) and 'So'. To the right of the main content is a sidebar titled 'SIGNED IN AS TUX' containing links for Profile, Starred, Settings (which has a cursor over it), Help, Site Administration, and Sign Out.

Wählen Sie hier *SSH / GPG Keys* und klicken unter *Manage SSH Keys* auf *Add Key*:



The screenshot shows the GitHub user profile page with the 'SSH / GPG Keys' tab selected. A large 'Add Key' button is visible in the top right corner of the main content area. Below it, a message states: 'These public SSH keys are associated with your account. The corresponding private keys allow full access to your repositories.' At the bottom of the page, there is a 'Manage GPG Keys' section with its own 'Add Key' button, a message about GPG keys, and a link to GitHub's 'about GPG' guide.

## 18 Ansible im Team

Fügen Sie anschließend in dem Feld *Content* den öffentlichen Schlüssel aus Ihrer Testumgebung ein und bestätigen Sie mit einem Klick auf *Add Key*:

The screenshot shows the GitHub user interface for managing SSH keys. The top navigation bar includes links for Dashboard, Issues, Pull Requests, Explore, Profile, Account, Security, Applications, **SSH / GPG Keys** (which is underlined), Organizations, and Repositories. Below this, a section titled "Manage SSH Keys" contains a button labeled "Add Key". A note below states: "These public SSH keys are associated with your account. The corresponding private keys allow full access to your repositories." A link provides help on creating SSH keys. The main form for adding a new key has fields for "Key Name" (set to "root@controller") and "Content" (containing a long public RSA key). A green "Add Key" button is at the bottom of this form.

Wechseln Sie nun zum *Dashboard*, klicken Sie hier auf das blaue Plus und erstellen Sie ein neues Repository:

The screenshot shows the 'New Repository' dialog box on a Gitea interface. At the top, there are navigation links: Dashboard, Issues, Pull Requests, Explore, and a user icon. Below the title 'New Repository', the 'Owner' field is set to 'tux'. The 'Repository Name' field contains 'ansible\_training'. A note below says 'Good repository names use short, memorable and unique keywords.' Under 'Visibility', there is an unchecked checkbox for 'Make Repository Private'. In the 'Description' section, a text area contains the placeholder 'Here we load everything from /root/code from the ansible training.' Below this, there are fields for '.gitignore' (with a 'Select .gitignore templates...' button) and 'License' (with a 'Select a license file...' button). A note above these fields reads 'Persönliches Exemplar von peter.wohlfarth@justiz.thueringen.de'. Under 'README', it says 'Default'. At the bottom, there is an unchecked checkbox for 'Initialize Repository (Adds .gitignore, License and README)' and two buttons: 'Create Repository' (green) and 'Cancel'.

Nun haben Sie ein Repository `ansible_test` in Ihrem Gitea angelegt. Im nächsten Schritt müssen Sie Ihren Ansible-Code aus `/root/code` in das Repository hochladen.



# Ansible-Code zum Gitea Repository hinzufügen

- ① Wechseln Sie in das Verzeichnis /root/code.
- ② Gehen Sie wie folgt vor:
  - ① Erstellen Sie ein Repository.
  - ② Fügen Sie den Inhalt von /root/code zu Ihrem Git-Index hinzu.
  - ③ „Committen“ Sie Ihre Änderungen.
  - ④ Verknüpfen Sie Ihr Repository mit Gitea.
  - ⑤ „Pushen“ Sie die Daten in das Repository im Gitea.

## Ein lokales Git-Repository anlegen

Nachdem die Gitea-Instanz eingerichtet ist und dort ein Repository für Ansible angelegt wurde, könne Sie Ihren lokalen Ansible-Code dem Repository hinzufügen. Wechseln Sie dafür in das Verzeichnis /root/code und führen Sie dort git init aus:

```
root@controller code]# git init
Initialized empty Git repository in /root/code/.git/
```

Fügen Sie anschließend alle Dateien in dem Verzeichnis Ihrem Index hinzu und machen Sie einen *commit* für alle Dateien:

```
root@controller code]# git add .
root@controller code]# git commit -ma "first commit"
```

Fügen Sie nun das Repository aus Gitea hinzu:

```
root@controller code]# git remote add origin
  ↳ git@git:tux/ansible_training.git
```

Nun können Sie alle Dateien aus dem Verzeichnis in Ihr Repository im Gitea *pushen*:

```
root@controller code]# git push -u origin master
The authenticity of host 'git (172.20.1.21)' can't be established.
ECDSA key fingerprint is
    ↗ SHA256:zRvyphQ/IBtQZkvVlmeEtbV3Zi1dbGYnRMSNjxfB1CI.
ECDSA key fingerprint is
    ↗ MD5:d7:86:3f:9f:4b:4b:b7:09:db:c4:c0:e4:11:9e:18:33.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'git,172.20.1.21' (ECDSA) to the list of
    ↗ known hosts.
Counting objects: 394, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (302/302), done.
Writing objects: 100% (394/394), 572.87 KiB | 22.00 KiB/s, done.
Total 394 (delta 42), reused 0 (delta 0)
remote: Resolving deltas: 100% (42/42), done.
To git@git:tux/ansible_training.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

tux / ansible\_training

Code Issues Pull Requests Releases Wiki Activity Settings

1 Commit 1 Branch

Branch: master New Pull Request New File Upload File HTTP SSH git@git:tux/ansible\_training.git

Tux Penguin	2223e31364	first commit	8 minutes ago
aufgaben		first commit	8 minutes ago
blocks		first commit	8 minutes ago
delegation		first commit	8 minutes ago
dynamic_inventory		first commit	8 minutes ago
foobooks		first commit	8 minutes ago
git		first commit	8 minutes ago
group_vars		first commit	8 minutes ago
host_vars		first commit	8 minutes ago
jinja2_examples		first commit	8 minutes ago
lookup		first commit	8 minutes ago
nested		first commit	8 minutes ago

## 18 Ansible im Team

Von nun an können Sie per Browser unter `http://git:3000/tux/ansible_training` sowie aus AWX auf die Daten zugreifen.

## 18.4 Ansible mit AWX



# Ansible mit AWX

- Ansible-Code aus verschiedenen Quellen importieren
- Ansible aus AWX ausführen
- Aufgaben automatisieren
- Zugriffs- und Ausführungsrechte verwalten

Bei AWX handelt es sich um die Upstreamversion von Red Hat's Ansible Tower. Mit Ansible Tower wie auch mit AWX steht Ihnen ein Werkzeug zur Verfügung, mit dem Sie Ansible-Code aus verschiedenen Quellen verwalten und ausführen können. Nachdem AWX eingerichtet ist, können Sie Aufgaben automatisieren und Ansible von Nutzern ausführen lassen, ohne dass diese eigene Zugriffsrechte auf den Zielsystemen bekommen müssten. AWX lädt den Ansible-Code aus verschiedenen Quellen und speichert Inventories wie auch Zugangsdaten wie Passwörter, Benutzer und Keys. Zudem findet die Administration der Projekte in einer grafischen Weboberfläche statt, sodass es nicht nötig ist, Ansible selbst auf der Kommandozeile zu bedienen. Durch diese Eigenschaften – das automatische Laden des Codes, die Verwaltung der Zugriffsrechte sowie das Arbeiten in der grafischen Oberfläche – bietet sich die Arbeit mit AWX bzw. Ansible Tower für die Arbeit in Teams an. Eine Entwicklungsabteilung kann den Ansible-Code entwickeln und mit AWX sogar automatisiert testen. Administratoren und andere Anwender können den Code jedoch in dem ihnen zugewiesenen Umfang per Klick selber ausführen und damit unabhängig arbeiten.

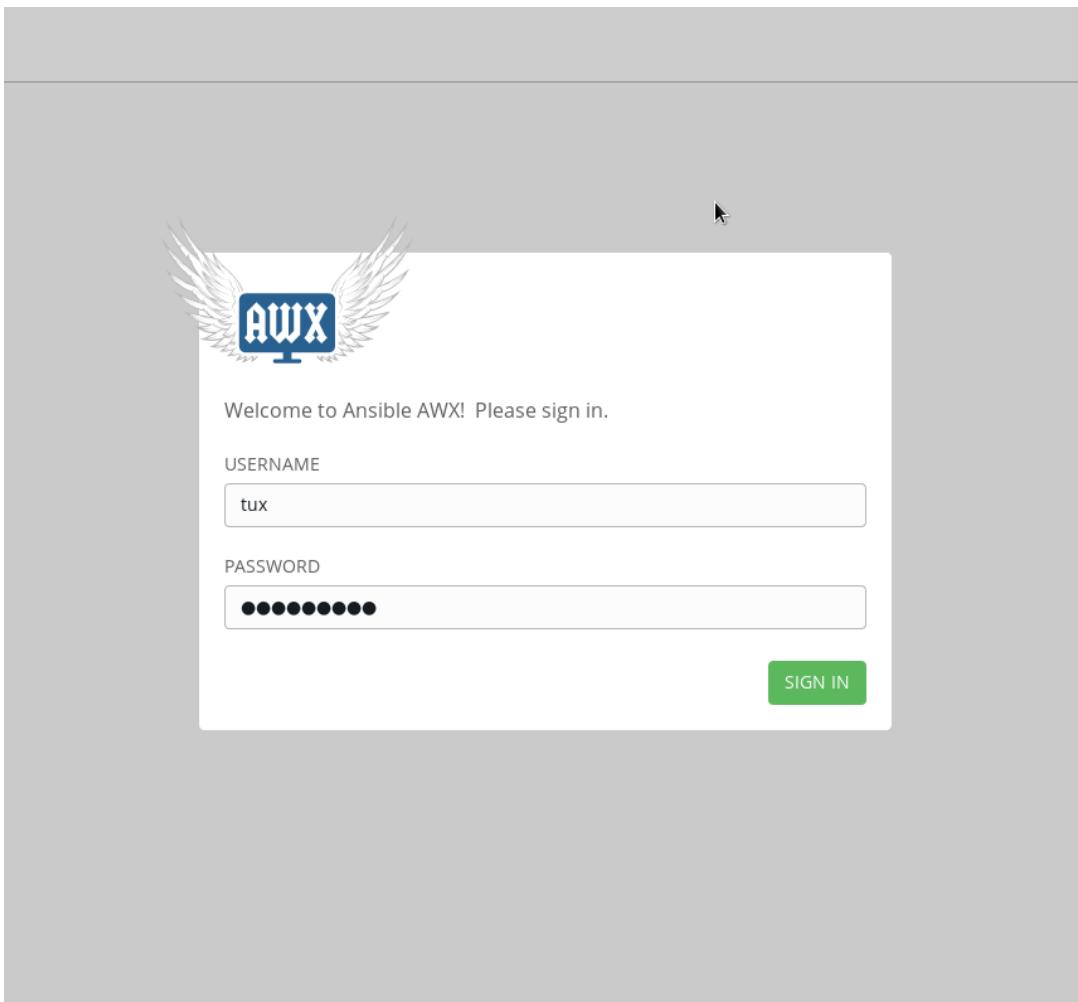
## Arbeiten mit AWX

Diese Schulungsumgebung verfügt über eine laufende AWX-Umgebung. Im Folgenden soll der bisherige Ansible-Code, den Sie im Rahmen dieser Schulung verfasst haben, aus dem Gitea in Ihre AWX-Umgebung geladen werden, damit Sie diesen aus AWX heraus ausführen können.

Öffnen Sie dafür die Url:

<http://localhost:8052>

in einem Webbrowser und loggen Sie sich mit dem *Username* tux und dem *Password* b1systems ein.



## AWX – Organisation hinzufügen

Zunächst müssen Sie AWX eine neue Organisation hinzufügen. Wählen Sie hierfür den Punkt *Organizations* aus dem Menü an der linken Seite aus und klicken Sie dann auf das grüne Plus, um eine neue Organisation zu erstellen:

The screenshot shows the AWX web interface. The left sidebar has a dark theme with white icons and text. It includes sections for VIEWS (Dashboard, Jobs, Schedules, My View), RESOURCES (Templates, Credentials, Projects, Inventories, Inventory Scripts), ACCESS (Organizations, Users, Teams), and ADMINISTRATION (Credential Types, Notifications, Management jobs). The 'Organizations' item under ACCESS is highlighted with a blue background. The main content area is titled 'ORGANIZATIONS' and shows a single organization named 'Default'. Below the title is a search bar and a 'KEY' button. The 'Default' organization card displays the following data:

- 0 USERS
- 0 TEAMS
- 1 INVEN...
- 1 PROJE...
- 1 JOB TE...
- 0 ADMINS

A message 'ITEMS 1 - 1' is at the bottom right of the card. In the top right corner of the main content area, there is a dark button with a green plus sign icon and the text 'Create a new organization'.

## 18 Ansible im Team

Tragen Sie als Namen der Organisation B1 Ansible Training ein und bestätigen Sie mit SAVE:

The screenshot shows the AWX web interface. On the left is a dark sidebar with a logo at the top and a navigation menu containing:

- VIEWS: Dashboard, Jobs, Schedules, My View.
- RESOURCES: Templates, Credentials, Projects, Inventories, Inventory Scripts.
- ACCESS: Organizations (highlighted in blue), Users, Teams.
- ADMINISTRATION: Credential Types, Notifications, Management Jobs.

The main content area has a header "ORGANIZATIONS / CREATE ORGANIZATION". A modal window titled "NEW ORGANIZATION" is open, showing three tabs: DETAILS (selected), USERS, and PERMISSIONS. The "NAME" field contains "B1 Ansible Training" and the "DESCRIPTION" field contains "Our first Organization". Below these fields is a search bar labeled "INSTANCE GROUPS". At the bottom right of the modal are "CANCEL" and "SAVE" buttons, with "SAVE" being highlighted in green. Below the modal is another section titled "ORGANIZATIONS" with a count of 1. It shows a table with one row labeled "Default" containing:

0	USERS	0	TEAMS
1	INVEN...	1	PROJE...
1	JOB TE...	0	ADMINS

A green "+" button is located to the right of the table.

## AWX – Nutzer anlegen

Nun können Sie der Organisation einen neuen Benutzer hinzufügen. Wählen Sie hierfür aus dem Menü an der linken Seite den Punkt *Users* aus und klicken Sie dort auf das grüne Kreuz um einen neuen Benutzer anzulegen:

The screenshot shows the AWX web interface. The left sidebar contains a navigation menu with sections like Views, Resources, Access, and Administration. The 'Users' option under 'Access' is currently selected. The main content area displays a table of users. There is one entry in the table: 'tux'. A prominent 'Create a new user' button with a green plus sign is located in the top right of the user list area.

## 18 Ansible im Team

Tragen Sie im Folgenden den *USERNAME* b1\_user und das *PASSWORD* b1s ein. Wählen Sie ferner unter *ORGANIZATION* die soeben angelegte Organisation B1 Ansible Training aus und tragen Sie unter *EMAIL* b1mail@localdomain.de ein. Bestätigen Sie anschließend Ihre Eingaben mit **SAVE**:

The screenshot shows the AUX application interface. The left sidebar contains navigation links for Views (Dashboard, Jobs, Schedules, My View), Resources (Templates, Credentials, Projects, Inventories, Inventory Scripts), Access (Organizations, Users, Teams), Administration (Credential Types, Notifications, Management Tasks), and a search bar. The 'Users' link in the sidebar is currently selected.

The main area displays a 'CREATE USER' form titled 'NEW USER'. The 'DETAILS' tab is active. The form fields are as follows:

- FIRST NAME: b1
- LAST NAME: user
- \* ORGANIZATION: B1 Ansible Training
- \* EMAIL: b1mail@localdomain.de
- \* USERNAME: b1\_user
- \* PASSWORD: (redacted)
- \* CONFIRM PASSWORD: (redacted)
- USER TYPE: Normal User

At the bottom right of the form are 'CANCEL' and 'SAVE' buttons. The 'SAVE' button is highlighted with a green background and a hand cursor icon.

Below the form, there is a table titled 'USERS' with one entry. The table has columns: USERNAME, FIRST NAME, LAST NAME, and ACTIONS. The 'ACTIONS' column for the single entry contains a green '+' button.

## AWX – Credentials hinterlegen

AWX ist in der Lage, Ansible-Code aus mehreren verschiedenen Quellen zu importieren und auf verschiedenen Zielumgebungen anzuwenden. Damit AWX den Code zum einen holen und zum anderen ausführen kann, müssen ggf. die Zugangsdaten in AWX hinterlegt werden. Wenn der Code z. B. in einer nicht-öffentlichen Git-Struktur liegt, so muss ein Key hinterlegt werden, mit dem AWX Zugang zu dem Git-Repository erhält. Zudem muss ein Key hinterlegt werden, mit dem AWX auf Zielsysteme zugreifen und dort Ansible ausführen darf.

Zugangsdaten werden in AWX unter dem Menüpunkt *Credentials* verwaltet. Um neue *Credentials* hinzuzufügen, klicken Sie auf den entsprechenden Menüpunkt im linken Menü und dort dann auf das grüne Plus:

NAME	KIND	OWNERS	ACTIONS
Demo Credential	Machine	tux	

ITEMS 1 - 1

Zunächst soll der Key hinterlegt werden, mit dem der Ansible-Code aus dem Gitea geholt wird. Hierfür benötigt AWX den *CREDENTIAL TYPE* Source Control. Als Namen können Sie SSH Key for Gitea eintragen und als Organisation wählen Sie B1 Ansible Training aus. In das Feld *SCM PRIVATE KEY* kopieren Sie Ihren privaten SSH-Schlüssel aus der Testumgebung, dessen öffentlichen Teil Sie bereits in Gitea hinterlegt haben. Abschließend klicken Sie auf *SAVE*:

The screenshot shows the AWX web interface. On the left is a dark sidebar with a navigation menu:

- Views: Dashboard, Jobs, Schedules, My View.
- Resources: Templates, Credentials.
- Access: Projects, Inventories, Inventory Scripts.
- Administration: Organizations, Users, Teams.
- Credential Types.
- Notifications.
- Management Jobs.

The main area is titled "CREDENTIALS / CREATE CREDENTIAL". A modal window is open for "NEW CREDENTIAL".

**DETAILS** tab is selected. Fields filled in:

- \* NAME: SSH Key for Gitea
- DESCRIPTION: this is the ssh key for tux
- ORGANIZATION: B1 Ansible Training

**PERMISSIONS** tab is also present but empty.

**TYPE DETAILS** section contains fields for:

- BENUTZERNAME: An empty search input field.
- PASSWORT: An empty search input field with an eye icon.

Below these fields is a note: "SCM PRIVATE KEY HINT: Drag and drop private file on the field below." A large text area contains a public RSA key:

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpATBAAKCAQEApI3SnETNNoUKPdflkp2vFAujGevg4ql0u15CSLQHTRZ+Ntszo
ByVurRxzqEY+ZFGUIkc/K1AZz8a08JUV/iBw+LOxmcZkzq2wPg5dTE62302wJ5gZ
+DNxGntzy+Zhj0AgolckUFP0qCixGDqAumCg+f44CS/DfcJ995CpNapS41K0Gb+
u76pDgQ4Khc9kwCPa0CwAgtbsf7b08narcfwnU96HWGMyXnr0U0AdmPgz5x32JE
0mlNaTyE+ZSB93YBBrh+dq1/14ySTnntKEiRMWrT/vYAnsY5F3p27+69NhKu71
i7AIIHJHf0Vn1dRtKya2nDp4u4aMqAqXrPArXWlWk7ar
```

Below the key is a note: "Personliches Exemplar von peter.wohlfarth@justiz.thueringen.de"

**PRIVATE KEY PASSPHRASE** field is empty.

At the bottom right of the modal are two buttons: **CANCEL** and **SAVE**, with **SAVE** being highlighted.

Um den Key für die Zielsysteme einzurichten, wiederholen Sie den Vorgang und klicken zunächst auf *Credentials* und danach auf das grüne Kreuz. Hier tragen Sie nun als Namen **SSH Key for Ansible** ein, wählen erneut **B1 Ansible Training** als Organisation aus und diesmal den **CREIDENTIAL TYPE** Maschine. Unter *Type Details* tragen Sie den Benutzernamen **root** ein und kopieren Ihren privaten SSH-Schlüssel in das Feld **SSH PRIVATE KEY**. Bestätigen Sie auch diesen Vorgang mit **SAVE**:

**NEW CREDENTIAL**

**DETAILS**    **PERMISSIONS**

\* NAME ?    DESCRIPTION ?    ORGANIZATION ?  
SSH Key for Ansible    The key to run ansible    B1 Ansible Training

\* CREDENTIAL TYPE ?  
Machine

**TYPE DETAILS**

BENUTZERNAME    PASSWORT     Prompt on launch  
root

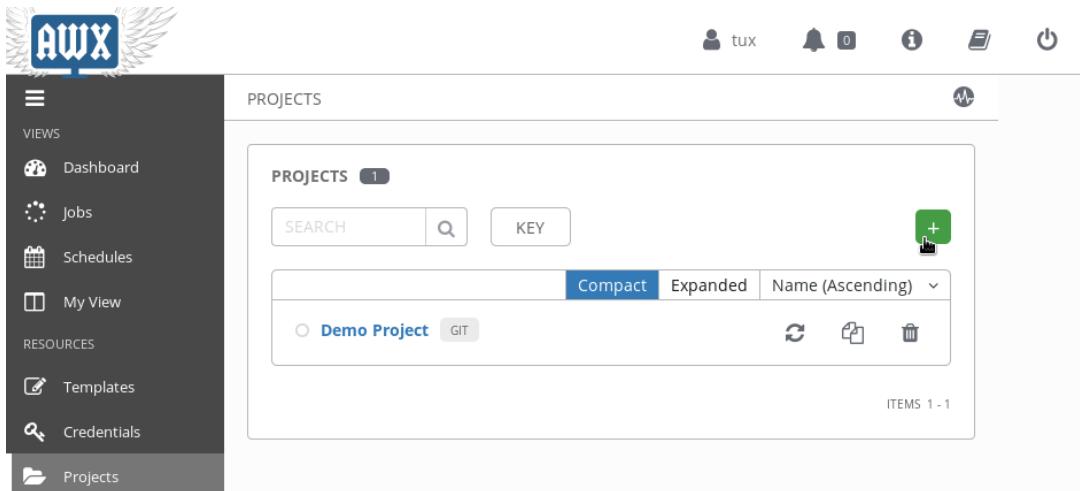
SSH PRIVATE KEY    HINT: Drag and drop private file on the field below.

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAwQDz7QfKrBxZ+b4vVH3sPu46gP6m1Wx+1D08Mj6oaldeG7U5M/+Jk/MDJ3
HeCdkHwKBgQC/N/DcGPnElJuhr4ebZ5RvLZMBGp/Qap/Pf3yGSwjVFn6pv8suyIjN
t62jQr8bSQZ6/k2aAMSt+XYGyapD4zIn6089LHEXj8jlRnNEyyffqCJE4aNveo
R3wvDkJowXT3DRp4ipn8ZhIYvjBwx/w3dSQuOcDbxFo/enIYSGxLyg==
-----END RSA PRIVATE KEY-----
```

SIGNED SSH CERTIFICATE    HINT: Drag and drop private file on the field below.

### AWX – Projekt anlegen

AWX verwaltet den Ansible-Code in Projekten. Damit der Ansible-Code aus dem Gitea in AWX geladen werden kann, müssen Sie zunächst ein Projekt dafür anlegen. Wählen Sie hierfür aus dem Menü links den Punkt *Projects* aus und klicken Sie anschließend auf das grüne Kreuz:



Im nächsten Schritt müssen Sie die Quelle für den Ansible-Code für das Projekt angeben. Tragen Sie zunächst den Namen Ansible for B1 ein und geben Sie als Organisation B1 Ansible Training an. Als SCM TYPE geben Sie Git an. Unter den SOURCE DETAILS geben Sie als SCM URL git@git:tux/ansible\_training.git an, die URL des Repositories aus Gitea. Unter SCM CREDENTIAL wählen Sie den SSH KEY for Gitea. Bestätigen Sie mit SAVE:

The screenshot shows the AWX interface with the following details:

- Project Details:**
  - Name: Ansible for B1
  - Description: All the things from /root/i
  - Organization: B1 Ansible Training
  - SCM Type: Git
  - SCM URL: git@git:tux/ansible\_traini
  - SCM Branch/Tag/Commit: master
  - SCM Refspec: (empty)
  - SCM Credential: SSH Key for Gitea
  - SCM Update Options:
    - CLEAN
    - DELETE ON UPDATE
    - UPDATE REVISION ON LAUNCH
    - ALLOW BRANCH OVERRIDE
- Buttons:** CANCEL and SAVE (highlighted in green)

## 18 Ansible im Team

Sobald Sie auf *SAVE* geklickt haben, beginnt AWX damit, einen *Job* zu erstellen und den Code aus dem Repository zu laden. Unter dem Punkt *Jobs* im linken Menü können Sie diesen Job einsehen. Mit einem Klick auf die Rakete können Sie den Job, etwa nach Code-Änderungen, erneut ausführen:

The screenshot shows the AWX web interface. On the left is a dark sidebar with a navigation menu. The 'Jobs' item in the 'JOBS' section is highlighted with a blue bar at the top. The main content area is titled 'JOBS' and shows a single job entry: '1 - Ansible for B1 SCM Update'. A small green circle icon next to the job name indicates it was successful. Below the job entry are three buttons: 'Compact', 'Expanded', and 'Finish Time (Descending)'. At the bottom right of the content area, it says 'ITEMS 1 - 1'. The top right of the screen shows user information ('tux') and several small icons.

Klicken Sie nun auf den Jobnamen oder den grünen Punkt davor, so können Sie die Details des Jobs sowie dessen Verlauf einsehen:

The screenshot shows the AWX interface with a successful job named "Ansible for B1". The job details include:

- STATUS:** Successful
- STARTED:** 25.10.2019 15:03:18
- FINISHED:** 25.10.2019 15:03:31
- JOB TYPE:** Check
- LAUNCHED BY:** tux
- PROJECT:** Ansible for B1
- BRANCH:** master
- REVISION:** 2223e31
- CREDENTIAL:** SSH Key for Gitea
- EXECUTION NODE:** awx.training.b1-systems.de
- INSTANCE GROUP:** tower

The right pane shows the Ansible play output:

```

Ansible for B1
PLAYS 2 TASKS 17 HOSTS 1 ELAPSED 00:00:12
[...]
56 skipping: [localhost] => {"changed": false, "skip_reason": "Conditional result was False"}
57
58 TASK [fetch galaxy collections from collections/requirements.yml] 15:03:31
*****
59 skipping: [localhost] => {"changed": false, "skip_reason": "Conditional result was False"}
60
61 PLAY RECAP
*****
62 localhost, : ok=3   changed=1  unreachable=0   failed=0
skipped=14    rescued=0    ignored=0
63

```

Hier ist zu sehen, dass der Job erfolgreich ausführt wurde und der Ansible-Code des Repositories in AWX geladen wurde. In der Detailansicht ist zudem zu erkennen, welche Elemente genau geladen wurden:

**Ansible for B1**  
 PLAYS 2 TASKS 17 HOSTS 1 ELAPSED 00:00:12

Im Folgenden muss der geladene Ansible-Code angewendet werden.

## AWX – Inventory anlegen

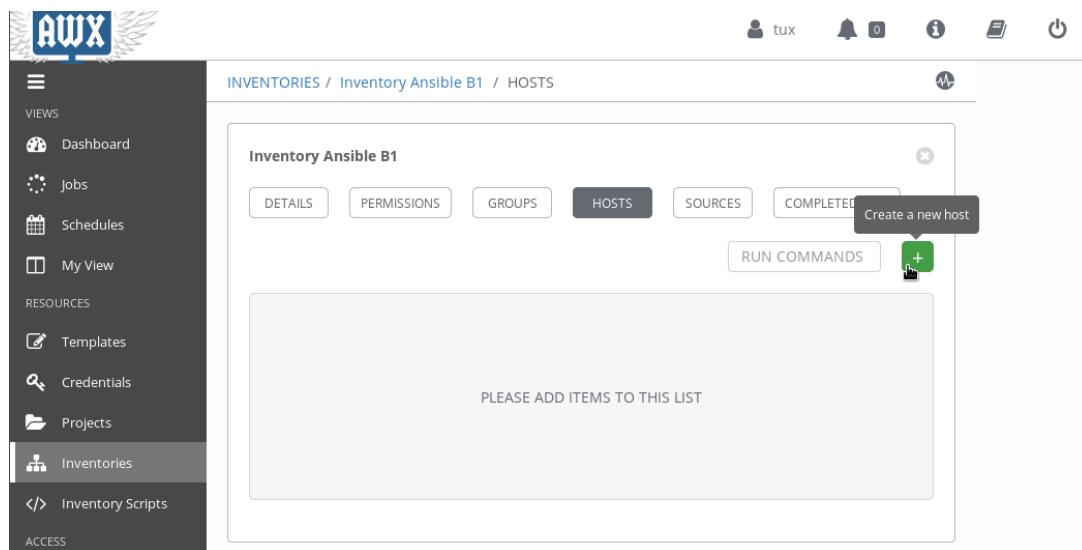
Damit der Ansible-Code von AWX ausgeführt werden kann, muss zunächst ein Inventory für die Zielsysteme erstellt werden. Wählen Sie dafür aus dem linken Menü den Punkt *Inventories* aus und klicken Sie danach auf das grüne Kreuz, um ein neues Inventory hinzuzufügen. Wählen Sie den Punkt *Inventory* aus:

The screenshot shows the AWX web interface. On the left is a dark sidebar with navigation links: Dashboard, jobs, Schedules, My View, Templates, Credentials, Projects, and Inventories (which is currently selected). The main area is titled 'INVENTORIES'. It contains two tabs: 'INVENTORIES' (selected) and 'HOSTS'. Below the tabs are search and key input fields. A table lists inventories with columns for NAME, TYPE, and ORGANIZATION. One entry, 'Demo Inventory', is shown with a cloud icon, type 'Inventory', and organization 'Default'. To the right of the table is a green '+' button with the text 'Create a new inventory' above it. At the bottom right of the table area, it says 'ITEMS 1 - 1'.

Im Folgenden müssen Sie zunächst einen Namen für das anzulegende Inventory sowie die Organisation angeben. Tragen Sie `Inventory Ansible B1` als Namen ein und geben Sie `B1 Ansible Training` an. Bestätigen Sie mit *SAVE*:

The screenshot shows the 'CREATE INVENTORY' form. The sidebar on the left has 'Inventories' selected. The main form is titled 'NEW INVENTORY' and has several tabs: DETAILS (selected), PERMISSIONS, GROUPS, HOSTS, SOURCES, and COMPLETED JOBS. Under the DETAILS tab, there are fields for NAME ('Inventory Ansible B1'), DESCRIPTION ('the old Inventory'), and ORGANIZATION ('B1 Ansible Training'). There are also fields for INSIGHTS CREDENTIAL and INSTANCE GROUPS. Below these are sections for VARIABLES (with tabs for YAML and JSON) and ACCESS (with tabs for Organizations and Users). At the bottom right are 'CANCEL' and 'SAVE' buttons, with 'SAVE' being green.

Nachdem das Inventory angelegt ist, können Sie unter dem Menüpunkt *HOSTS* mit einem Klick auf das grüne Plus einzelne Hosts hinzufügen:



The screenshot shows the AWX web interface. On the left is a sidebar with sections for VIEWS (Dashboard, Jobs, Schedules, My View), RESOURCES (Templates, Credentials, Projects, Inventories, Inventory Scripts), and ACCESS. The 'Inventories' item under Resources is currently selected. The main area shows the 'INVENTORIES / Inventory Ansible B1 / HOSTS' view. At the top, there are tabs for DETAILS, PERMISSIONS, GROUPS, HOSTS (which is selected and highlighted in dark grey), SOURCES, and COMPLETED. Below these tabs is a button labeled 'Create a new host' with a green plus sign icon. A large central box contains the text 'PLEASE ADD ITEMS TO THIS LIST'. The top right of the interface has user information (tux), notifications, and other system icons.

Übertragen Sie die Hosts aus dem bisher in der Schulung verwendeten Inventory. Tragen Sie unter *HOST NAME* die Namen der Hosts ein und bestätigen Sie mit *SAVE*. Beachten Sie, dass das grüne Plus, nachdem Sie den ersten Host angelegt haben, auf der Seite nach unten versetzt wird:

The screenshot shows the AWX web interface. On the left is a dark sidebar with various navigation options: Dashboard, Jobs, Schedules, My View, Templates, Credentials, Projects, Inventories (which is selected), Inventory Scripts, Organizations, Users, Teams, Credential Types, Notifications, Management Jobs, Instance Groups, Applications, and Settings.

The main content area has a header "INVENTORIES / Inventory Ansible B1 / HOSTS / CREATE HOST". Below it is a "CREATE HOST" dialog box. The "DETAILS" tab is active, showing a field for "HOST NAME" with "centos2" entered. There is also a "DESCRIPTION" field and a "VARIABLES" section with tabs for "YAML" and "JSON". A large text area for YAML variables is empty. At the bottom of the dialog are "CANCEL" and "SAVE" buttons.

Below the dialog is a list titled "Inventory Ansible B1" with the "HOSTS" tab selected. It shows five hosts: centos1, suse1, suse2, ubuntu1, and ubuntu2. Each host has a checkbox, a status indicator, and a name. To the right of each host are edit and delete icons. Above the host list are buttons for "COMPLETED JOBS", "SEARCH", "KEY", "RUN COMMANDS", and a green "+" button with a cursor icon pointing to it. A tooltip "Create a new host" is shown above the "+" button. At the bottom of the list is a "ITEMS 1 - 5" message.

Damit die Gruppe `webserver` aus dem alten Inventory abgebildet wird, muss diese erst erstellt werden. Wählen Sie den Menüpunkt `GROUPS` aus und klicken Sie anschließend auf das grüne Kreuz, um eine neue Gruppe anzulegen:

The screenshot shows the AWX web interface. On the left is a dark sidebar with various navigation options like Dashboard, Jobs, Schedules, My View, Templates, Credentials, Projects, Inventories (which is currently selected), Inventory Scripts, and Organizations. The main area is titled 'INVENTORIES / Inventory Ansible B1 / ALL GROUPS'. It has tabs for DETAILS, PERMISSIONS, GROUPS (which is active and highlighted in dark grey), HOSTS, and SOURCES. Below these tabs is a 'COMPLETED JOBS' section. Under the GROUPS tab, there are buttons for 'ALL GROUPS' (which is active and highlighted in blue), 'ROOT GROUPS', and 'RUN COMMANDS'. To the right of these buttons is a green '+' button with a white plus sign, which is also highlighted with a red box. Below the buttons is a large text input field containing 'PLEASE ADD ITEMS TO THIS LIST'.

Tragen Sie den Namen `webserver` ein und bestätigen Sie mit `SAVE`:

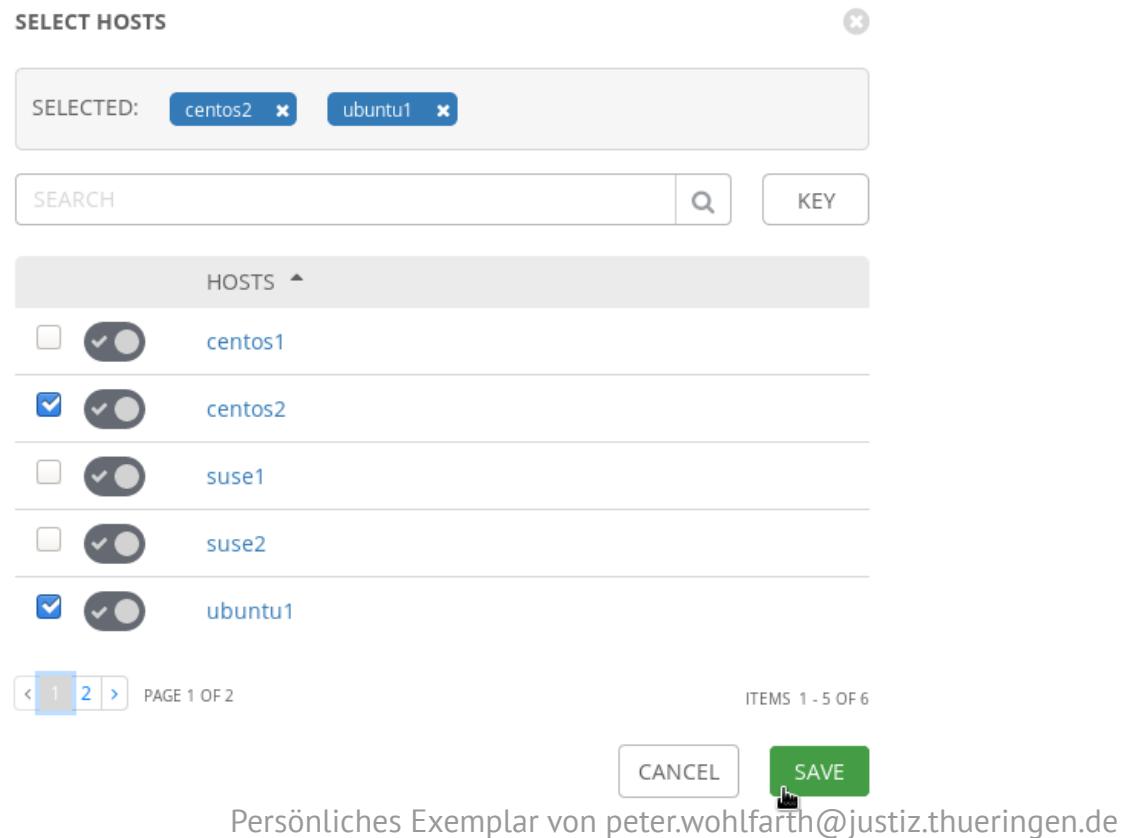
The screenshot shows the 'CREATE GROUP' dialog box. The sidebar on the left is identical to the previous screenshot. The main area is titled 'INVENTORIES / Inventory Ansible B1 / ALL GROUPS / CREATE GROUP'. It has tabs for DETAILS, GROUPS (which is active and highlighted in dark grey), and HOSTS. Below these tabs is a form with a required field 'NAME' containing 'webserver' and an optional 'DESCRIPTION' field which is empty. Below the form is a 'VARIABLES' section with tabs for 'YAML' (which is active and highlighted in blue) and 'JSON'. At the bottom of the dialog are 'CANCEL' and 'SAVE' buttons, with 'SAVE' being highlighted with a red box.

## 18 Ansible im Team

Als nächstes müssen Sie der Gruppe Hosts hinzufügen. Wählen Sie in der Ansicht der Gruppe *webserver* den Punkt *Hosts* aus klicken Sie dort auf das grüne Plus, um der Gruppe Hosts hinzuzufügen. Wählen Sie hier *Existing Host*:

The screenshot shows the AWX inventory management interface. On the left is a sidebar with navigation links: VIEWS (Dashboard, Jobs, Schedules, My View), RESOURCES (Templates, Credentials, Projects, Inventories, Inventory Scripts), and ACCESS. The main area shows the 'INVENTORIES / Inventory Ansible / ALL GROUPS / webserver / ASSOCIATED HOSTS...' path. A modal window titled 'webserver' is open, showing three tabs: DETAILS, GROUPS, and HOSTS (which is selected). In the bottom right corner of the modal, there is a green '+' button. A tooltip for this button says 'Add a host'. A dropdown menu from this button contains two options: 'Existing Host' (highlighted with a red box) and 'New Host'. Below the modal, a message says 'PLEASE ADD ITEMS TO THIS LIST'.

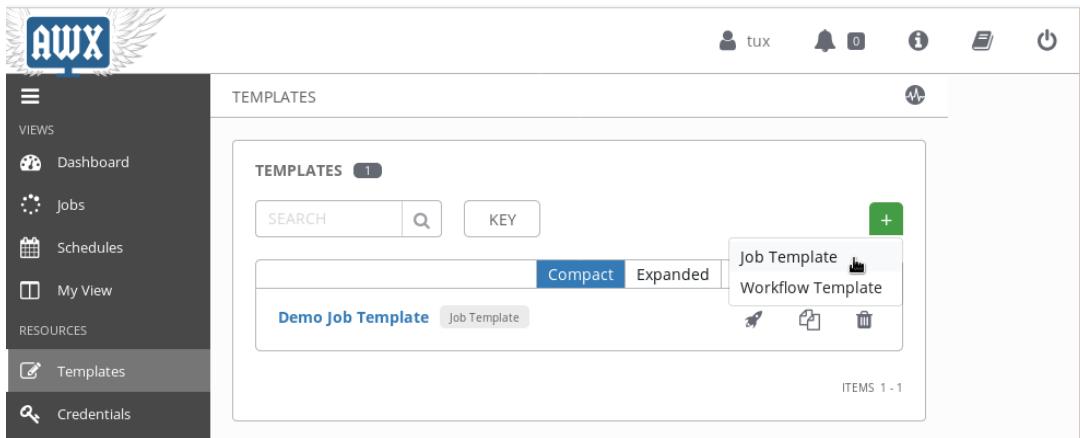
Im nun erscheinenden Dialogfenster wählen Sie die Hosts `centos2` und `ubuntu1` aus und fügen Sie mit einem Klick auf *SAVE* der Gruppe hinzu:



Jetzt ist Ihr AWX-Inventory mit jenem identisch, welches Sie zuvor direkt mit Ansible verwendet haben. Sie können nun Code auf den im Inventory eingetragenen Hosts ausführen.

### AWX – Templates anlegen

Damit AWX Ansible-Code ausführen kann, muss dieser in *Job Templates* eingebunden werden, welche dann ausgeführt werden können. Um ein *Template* anzulegen, klicken Sie im linken Menü auf den Punkt *Templates* und anschließend auf das grüne Plus und wählen dort *Job Template*, um ein neues *Template* anzulegen:



The screenshot shows the AWX web interface. The left sidebar has a dark theme with white icons and text. It includes sections for 'VIEWS' (Dashboard, Jobs, Schedules, My View) and 'RESOURCES' (Templates, Credentials). The 'TEMPLATES' option is highlighted with a blue background. The main content area is titled 'TEMPLATES' and shows a single item: 'Demo Job Template' (Job Template). Above the list are search and key buttons. To the right of the list are three icons: a pencil for edit, a copy symbol, and a trash can for delete. At the bottom right of the list area, it says 'ITEMS 1 - 1'. The top right of the screen shows user information ('tux'), notifications (0), and other system icons.

In dem Formular *NEW JOB TEMPLATE* vergeben Sie den Namen Apache2 role from B1 training und wählen als Inventory Inventory Ansible B1 aus. Sobald Sie unter *PROJECT* Ansible for B1 ausgewählt haben, erscheint ein weiteres Feld *PLAYBOOK*, in dem Sie aus allen aus dem Gitea in das Projekt Ansible for B1 geladenen Playbooks das Playbook apache\_role.yml auswählen. Unter *CREDENTIALS* geben Sie den SSH Key for Ansible an, damit AWX auf die Inventory-Hosts zugreifen kann. Speichern Sie das Template mit *SAVE* ab.

## 18 Ansible im Team

Klicken Sie nun erneut auf den Punkt *Templates* im linken Menü. Sie finden hier nun Ihr soeben erstelltes Template, welches die Rolle *apache2* für die Gruppe *webserver* ausführt, sobald es aktiviert wird. Um den Job zu starten, klicken Sie auf die Rakete:

The screenshot shows the AWX interface with a sidebar on the left containing links like Dashboard, Jobs, Schedules, My View, Templates (which is selected), Credentials, and Projects. The main area is titled 'TEMPLATES (2)' and lists two items: 'Apache2 role from B1 training' (Job Template) and 'Demo Job Template' (Job Template). Each item has a 'Start a job using this template' button represented by a blue rocket icon.

Die Rolle wird nun von AWX auf den Zielsystemen genauso ausgeführt wie zuvor von Ansible auf der Kommandozeile:

The screenshot shows the AWX interface with a sidebar on the left. The main area displays a job run titled 'JOBS / 2 - Apache2 role from B1 training'. The 'DETAILS' section shows the status as 'Successful' and provides information about the job template, launch, inventory, project, revision, playbook, credential, environment, execution node, and instance group. The 'LOG' section shows the command-line output of the job, which includes the creation of an SSH key, gathering facts, and applying the apache2 role to the 'suse2' host.

```
1 Identity added: /tmp/awx_2_8elatxqi/artifacts/2/ssh_key_data (/tmp/awx_2_8elatxqi/artifacts/2/ssh_key_data)
2
3 PLAY [this is the first part towards a role for a fully configured apache2 webserver] ***
4
5 TASK [Gathering Facts]
*****
6 ok: [suse2]
7
8 TASK [apache2 : include_vars]
*****
9 ok: [suse2]
10
11 TASK [apache2 : ensure that the "present" version of the apache2 package is there] ***
12 skipping: [suse2]
13
14 TASK [apache2 : ensure that the service is up and running]
*****
15 skipping: [suse2]
16
17 TASK [apache2 : include_tasks]
*****
18
19 PLAY RECAP
*****
20 suse2 : ok=2    changed=0    unreachable=0    failed=0    skipped=3    rescued=0    ignored=0
```

## 18.5 Aufgabenteil



### Aufgabenteil – Ansible mit AWX

- ① Weisen Sie dem Benutzer `b1_user` die nötigen Rechte zu, die Rolle `apache2` auszuführen. Loggen Sie sich als Nutzer `b1_user` ein und führen Sie die Rolle aus.
- ② Legen Sie in einem neuen Branch `devel` ein Playbook an, das für alle Hosts aus dem Inventory das Modul `ping` ausführt. *Pushen* Sie den neuen Branch in Ihr Repository in Gitea. Holen Sie nun die Daten aus dem neuen Branch in Ihre AWX-Instanz und führen Sie das Playbook aus.

1. Weisen Sie dem Benutzer `b1_user` die nötigen Rechte zu, die Rolle `apache2` auszuführen. Loggen Sie sich als Nutzer `b1_user` ein und führen Sie die Rolle aus.
2. Legen Sie in einem neuen Branch `devel` ein Playbook an, das für alle Hosts aus dem Inventory das Modul `ping` ausführt. *Pushen* Sie den neuen Branch in Ihr Repository in Gitea. Holen Sie nun die Daten aus dem neuen Branch in Ihre AWX-Instanz und führen Sie das Playbook aus.

## Musterlösung

- Weisen Sie dem Benutzer `b1_user` die nötigen Rechte zu, die Rolle `apache2` auszuführen. Loggen Sie sich als Nutzer `b1_user` ein und führen Sie die Rolle aus:

Wählen Sie zunächst im linken Menü den Punkt *Users* aus und klicken Sie dann auf das Stiftsymbol des Benutzers `b1_user`, um dessen Profil zu editieren:

The screenshot shows the AWX web interface. On the left is a dark sidebar with navigation links: Dashboard, Jobs, Schedules, My View, Templates, Credentials, Projects, and Inventories. The main area is titled 'USERS' and shows a list of users with 2 items. The first item is 'b1\_user' with first name 'b1' and last name 'user'. Below the list is a note 'tux'. At the top right of the user card is a 'Edit user' button, which is highlighted with a blue cursor icon.

Wählen Sie nun den Menüpunkt *PERMISSIONS* und klicken Sie auf das grüne Kreuz, um dem Benutzer Rechte zuzuweisen:

The screenshot shows the AWX interface with the 'Permissions' tab selected for the user 'b1\_user'. The sidebar on the left has 'Users' selected. The main area shows the 'PERMISSIONS' tab for 'b1\_user'. It includes a 'Grant Permission' button with a green plus sign and a table listing assigned permissions. The table has columns: NAME, TYPE, ROLE, and ACTIONS. It shows two entries: 'B1 Ansible Training' (Organization, Member) and 'Apache2 role from B1 training' (Job Template, Execute). Below this is another 'USERS' list for 'b1\_user'.

Wählen Sie hier unter dem Menüpunkt 1 *JOB TEMPLATES* und dort das Template *Apache2 role from B1 training* aus. Unter Punkt 2 wählen Sie als Umfang der Rechte *Execute* aus dem Dropdownmenü. Bestätigen Sie anschließend mit *SAVE*:

B1\_USER | ADD PERMISSIONS

1 Please select resources from the lists below.

JOB TEMPLATES     WORKFLOW TEMPLATES     PROJECTS     INVENTORIES     CREDENTIALS  
 ORGANIZATIONS

SEARCH  KEY

NAME ▾

Apache2 role from B1 training  
 Demo Job Template

ITEMS 1 - 2

---

2 Please assign roles to the selected resources

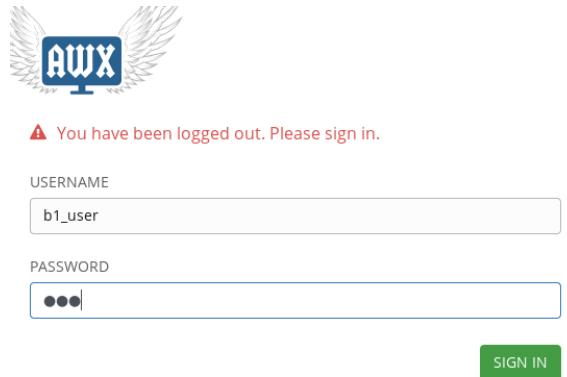
JOB TEMPLATES

Execute	KEY
Admin	
Execute	ACTIONS <input type="button" value="✖"/>
Read	

Add Permissions

## 18 Ansible im Team

Loggen Sie sich aus und erneut mit dem Benutzer `b1_user` und dem Passwort `b1s` ein:



Wenn Sie jetzt den Menüpunkt *Templates* aus dem linken Menü aufrufen, so sehen Sie das Template *Apache2 role from B1 training*. Mit einem Klick auf das Raketen-Symbol können Sie das Template ausführen:

The screenshot shows the AWX interface with the 'TEMPLATES' menu item selected in the sidebar. The main area displays a single template entry: 'Apache2 role from B1 training'. A context menu is open over this entry, with the option 'Start a job using this template' highlighted.

Das Template mit der Rolle *apache2* wird vom Nutzer `b1_user` ausgeführt:

The screenshot shows the AWX interface with the 'JOBS' menu item selected in the sidebar. A specific job entry is selected: '6 - Apache2 role from B1 training'. The job details show it was launched by 'b1\_user' and is in a 'Successful' state. The terminal pane on the right displays the execution log, which includes task logs and a play recap. The log output shows a task for 'apache2 : include\_tasks' and a play recap for 'suse2'.

2. Legen Sie in einem neuen Branch *devel* ein Playbook an, das für alle Hosts aus dem Inventory das Modul `ping` ausführt. Pushen Sie den neuen Branch in Ihr Repository in Gitea. Holen Sie nun die Daten aus dem neuen Branch in Ihre AWX-Instanz und führen Sie das Playbook aus:

Wechseln Sie in das Verzeichnis `/root/code` und erstellen Sie mit `git checkout -b devel` den neuen Branch `devel`:

```
# git checkout -b devel
M     git/ansible_test
Switched to a new branch 'devel'
```

Erstellen Sie nun das Playbook:

```
---
- name: this play should ping all hosts
  hosts: all
  tasks:
    - name: the actual ping task
      ansible.builtin.ping:
```

Fügen Sie das Playbook zu Ihrem Index hinzu und *committen* und *pushen* Sie es:

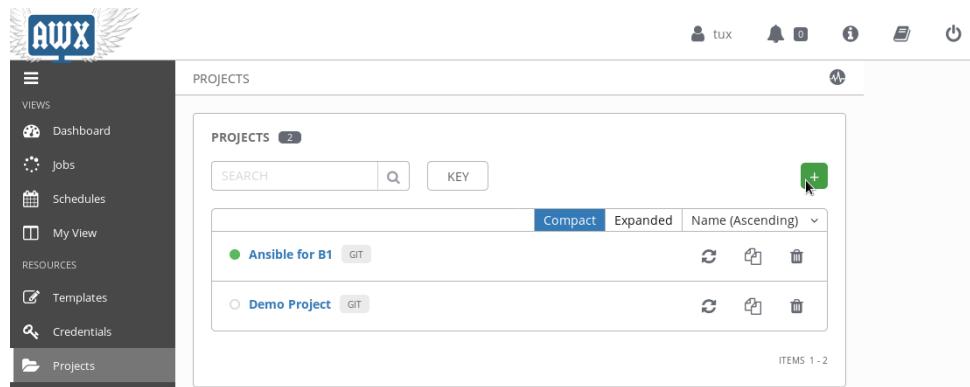
```
# git add awx_ping_all.yml

# git commit awx_ping_all.yml -m "the playbook for awx"
[devel 7a7e2f9] the playbook for awx
 1 file changed, 6 insertions(+)
 create mode 100644 awx_ping_all.yml

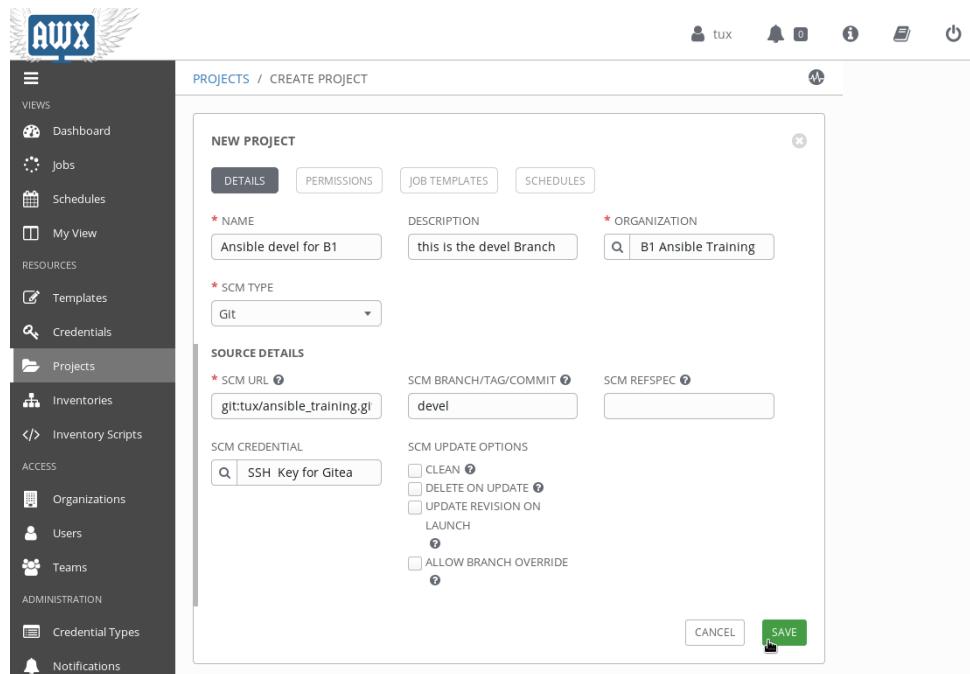
# git push -u origin devel
Counting objects: 4, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 372 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote:
remote: Create a new pull request for 'devel'!
remote:
remote:    ↗ http://git:3000/tux/ansible_training/compare/master...devel
remote:
To git@git:tux/ansible_training.git
 * [new branch]      devel -> devel
Branch devel set up to track remote branch devel from origin.
```

## 18 Ansible im Team

Damit Sie Ihren neuen Branch in AWX nutzen können, müssen Sie für diesen ein neues Projekt anlegen. Wählen Sie aus dem linken Menü den Punkt *Projects* und fügen Sie mit dem grünen Kreuz ein neues Projekt hinzu:



Tragen Sie hier einen Projektnamen und die Organisation B1 Ansible Training ein. Als *SCM TYPE* wählen Sie Git und die *SCM URL* ist die URL Ihres Gitea-Repositories, `git@git:tux/ansible_training.git`. Unter *SCM BRANCH/TAG/COMMIT* tragen Sie `devel` ein. Mit einem Klick auf *SAVE* bestätigen Sie:



Nachdem AWX die Daten aus dem Repository geladen hat, können Sie ein Template anlegen, in dem das Playbook ausgeführt werden kann. Wählen Sie hierfür zunächst den Menüpunkt Templates aus dem linken Menü aus, klicken Sie auf das grüne Kreuz und wählen Sie *Job Template*:

The screenshot shows the AWX web interface. On the left is a dark sidebar with a navigation menu. The 'TEMPLATES' item in the sidebar is highlighted with a grey background. The main content area has a light grey header bar with the title 'TEMPLATES'. Below this is a search bar with 'SEARCH' and a magnifying glass icon, and a 'KEY' button. To the right of the search bar is a green '+' button. Below the search bar are two tabs: 'Compact' (which is selected) and 'Expanded'. A tooltip for the 'Compact' tab says 'Job Template'. Below these tabs is a list of templates. The first template listed is 'Apache2 role from B1 training' (Job Template). To its right is a grid of small icons representing other templates. Below this grid is a row of three icons: a pencil, a copy symbol, and a trash can. At the bottom of the list is a footer bar with the text 'ITEMS 1 - 2'.

Hier vergeben Sie einen Namen, und wählen das *INVENTORY* Inventory Ansible B1 und das *PROJECT* Ansible devel for B1 aus. Als *PLAYBOOK* wählen Sie das Playbook, das Sie für diese Aufgabe geschrieben haben. Nun müssen Sie nur noch die *CREDENTIALS* SSH Key for Ansible auswählen und können das Template mit *SAVE* speichern:

The screenshot shows the AWX web interface for creating a new job template. The left sidebar contains navigation links for Views (Dashboard, Jobs, Schedules, My View), Resources (Templates, Credentials, Projects, Inventories, Inventory Scripts, Organizations, Users, Teams), Access (Management Jobs, Instance Groups, Applications, Settings), and Administration (Credential Types, Notifications). The main area is titled "TEMPLATES / CREATE JOB TEMPLATE" and "NEW JOB TEMPLATE". The "DETAILS" tab is selected. The form fields include:

- NAME:** The Playbook that Pings
- DESCRIPTION:** this playbook just pings all tl
- JOB TYPE:** Run
- INVENTORY:** Inventory Ansible B1
- PROJECT:** Ansible devel for B1
- PLAYBOOK:** awx\_ping\_all.yml
- CREDENTIALS:** SSH Key for Ans... (selected)
- FORKS:** 0
- LIMIT:** (empty)
- VERBOSITY:** 0 (Normal)
- JOB TAGS:** (empty)
- SKIP TAGS:** (empty)
- LABELS:** (empty)
- INSTANCE GROUPS:** (empty)
- JOB SLICING:** 1
- TIMEOUT:** 0
- SHOW CHANGES:** (switched off)
- OPTIONS:**
  - ENABLE PRIVILEGE ESCALATION
  - ENABLE PROVISIONING CALLBACKS
  - ENABLE WEBHOOK
  - ENABLE CONCURRENT JOBS
  - ENABLE FACT CACHE
- EXTRA VARIABLES:** (YAML editor with content: 1 ...)

At the bottom right are buttons for **LAUNCH**, **CANCEL**, and **SAVE**. A watermark "Personliches Exemplar von peter.wohlarth@justiz.thueringen.de" is overlaid across the center of the form.

Wählen Sie erneut den Menüpunkt *Templates* aus dem linken Menü aus und klicken Sie auf das Raketensymbol Ihres neu angelegten Templates, um dieses auszuführen:

Das Playbook wird nun ausgeführt:

```

PLAYS: 1   TASKS: 22   HOSTS: 6   ELAPSED: 00:00:14
=====
7 ok: [ubuntu1]
8 ok: [suse1]
9 ok: [centos2]
10 ok: [centos1]
11 ok: [suse2]
12
13 TASK [the actual ping task]
*****
14 ok: [ubuntu2]
15 ok: [centos1]
16 ok: [centos2]
17 ok: [suse1]
18 ok: [suse2]
19 ok: [ubuntu1]
20
21 PLAY RECAP
*****

```

## 18.6 Ansible Tower



# Ansible Tower

Ansible Tower bietet

- andere Installationsmöglichkeiten
- Releasezyklen
- Long-Term-Support

Bei Ansible Tower handelt es sich um das kommerzielle Produkt, das aus dem Upstreamprojekt AWX hervorgeht und von Red Hat vertrieben wird. Funktional wie strukturell unterscheiden sich AWX und Ansible Tower nicht voneinander. Jedoch folgt Ansible Tower anderen Installationsroutinen und lässt sich beispielsweise auch direkt auf dem *Bare Metal* (also nicht in einem Container) installieren. Ebenso folgt Ansible Tower in der Versionierung von Red Hat festgelegten Releasezyklen und Sie können für Ansible Tower Support von Red Hat in Anspruch nehmen.

Die grafische Benutzeroberfläche unterscheidet sich kaum von AWX:

The screenshot shows the Ansible Tower dashboard. On the left is a sidebar with navigation links: Views (Dashboard, Jobs, Schedules, My View), Resources (Templates, Credentials, Projects, Inventories), Access (Organizations, Users, Teams), Administration (Credential Types, Notifications, Management Jobs). The main area has a header "DASHBOARD" with user info (admin) and system icons. It features a summary box with counts: 2 HOSTS, 0 FAILED HOSTS, 2 INVENTORIES, 0 INVENTORY SYNC FAILURES, 3 PROJECTS, 1 PROJECT SYNC FAILURES. Below is a "JOB STATUS" chart showing the number of jobs over time from Sep 29 to Oct 29. A red line shows failed hosts peaking at 4 on Oct 13, and a green line shows successful hosts peaking at 7 on Oct 13. Filter options for Period (PAST MONTH), Job Type (ALL), and View (ALL) are present. To the right are two tables: "RECENTLY USED TEMPLATES" listing "second tes" and "first test" with activity icons, and "RECENT JOB RUNS" listing four entries with names, times, and status icons.

## Dashboard

The screenshot shows the Ansible Tower inventories section. The sidebar includes "Inventories" under Resources. The main area has a header "INVENTORIES" with user info (admin) and system icons. It features a search bar and a "Create a new inventory" button. A table lists inventories: "betacloud\_Inventory" (Inventory, B1 Ansible Tower) and "Demo Inventory" (Inventory, Default). Action columns provide edit, copy, and delete options. A footer indicates "ITEMS 1 - 2".

## Inventory-Abschnitt

## 18 Ansible im Team

The screenshot shows the Ansible Tower web interface. The left sidebar contains a navigation menu with sections like Views, Resources, ACCESS, and ADMINISTRATION. The 'Projects' section is currently selected. The main content area is titled 'PROJECTS' and shows two projects: 'Demo Project' and 'the gittea tea'. Each project has a green circular icon, a name, and a small 'GIT' badge. Below the projects is a status bar with 'ITEMS 1 - 2'.

### Projekt-Abschnitt

The screenshot shows the Ansible Tower web interface. The left sidebar contains a navigation menu with sections like Views, Resources, ACCESS, and ADMINISTRATION. The 'Credentials' section is currently selected. The main content area is titled 'CREDENTIALS' and shows two credentials: 'Demo Credential' and 'SSH KEY for the client'. Each credential has a grey circular icon, a name, a kind (Machine), and an owner (admin). A tooltip 'Create a new credential' is visible near the '+' button. Below the credentials is a status bar with 'ITEMS 1 - 2'.

### Abschnitt für die Credentials

## Job-Übersicht

```

1 Identity added: /tmp/awx_30_ck5v3krw/artifacts/3
0/ssh_key_data (/tmp/awx_30_ck5v3krw/artifacts/3
0/ssh_key_data)
2 SSH password:
3
4 PLAY [all]
*****
5
6 TASK [ping]
*****
7 ok: [centos]
8
9 PLAY RECAP
*****
10 centos : ok=1    changed=0
unreachable=0   failed=0    skipped=0   rescue
d=0    ignored=0
11

```

## Status eines ausgeführten Jobs