





# **Red Hat Enterprise Linux- Automatisierung mit Ansible**



**Red Hat Enterprise Linux 8.4 RH294**  
**Red Hat Enterprise Linux-Automatisierung mit Ansible**  
**Ausgabe 1 20210818**  
**Veröffentlicht 20210818**

Autoren: Trey Feagle, Hervé Quatremain, Dallas Spohn, Adolfo Vazquez,  
Morgan Weetman  
Editor: Philip Sweany, Seth Kenlon, Jeff Tyson, Nicole Muller, David  
O'Brien

Copyright © 2021 Red Hat, Inc.

The contents of this course and all its modules and related materials, including handouts to audience members, are  
Copyright © 2021 Red Hat, Inc.

No part of this publication may be stored in a retrieval system, transmitted or reproduced in any way, including, but  
not limited to, photocopy, photograph, magnetic, electronic or other record, without the prior written permission of  
Red Hat, Inc.

This instructional program, including all material provided herein, is supplied without any guarantees from Red Hat,  
Inc. Red Hat, Inc. assumes no liability for damages or legal action arising from the use or misuse of contents or details  
contained herein.

If you believe Red Hat training materials are being used, copied, or otherwise improperly distributed, please send  
email to [training@redhat.com](mailto:training@redhat.com) or phone toll-free (USA) +1 (866) 626-2994 or +1 (919) 754-3700.

Red Hat, Red Hat Enterprise Linux, the Red Hat logo, JBoss, OpenShift, Fedora, Hibernate, Ansible, CloudForms,  
RHCA, RHCE, RHCSA, Ceph, and Gluster are trademarks or registered trademarks of Red Hat, Inc. or its subsidiaries  
in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle and/or its affiliates.

XFS® is a registered trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or  
other countries.

MySQL® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent  
Node.js open source or commercial project.

The OpenStack® Word Mark and OpenStack Logo are either registered trademarks/service marks or trademarks/  
service marks of the OpenStack Foundation, in the United States and other countries and are used with the  
OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack  
Foundation or the OpenStack community.

All other trademarks are the property of their respective owners.

Mitwirkende: James Mighion, Alejandra Ramirez Palacios, Michael Phillips

Teile dieses Kurses basieren auf dem Ansible Lightbulb-Projekt. Das Material aus diesem Projekt  
ist ab <https://github.com/ansible/lightbulb> mit der MIT-Lizenz verfügbar.

<b>Dokumentkonventionen</b>	<b>vii</b>
	vii
<b>Einführung</b>	<b>ix</b>
Red Hat Enterprise Linux-Automatisierung mit Ansible .....	ix
Informationen zur Kursumgebung .....	xi
<b>1. Einführung in Ansible</b>	<b>1</b>
Automatisieren der Linux-Administration mit Ansible .....	2
Quiz: Automatisieren der Linux-Administration mit Ansible .....	9
Installieren von Ansible .....	11
Angeleitete Übung: Installieren von Ansible .....	16
Zusammenfassung .....	18
<b>2. Implementieren eines Ansible-Playbooks</b>	<b>19</b>
Erstellen eines Ansible-Inventars .....	21
Angeleitete Übung: Erstellen eines Ansible-Inventars .....	26
Verwalten von Ansible-Konfigurationsdateien .....	31
Angeleitete Übung: Verwalten von Ansible-Konfigurationsdateien .....	39
Ausführen von Ad-hoc-Befehlen .....	43
Angeleitete Übung: Ausführen von Ad-hoc-Befehlen .....	50
Schreiben und Ausführen von Playbooks .....	55
Angeleitete Übung: Schreiben und Ausführen von Playbooks .....	62
Implementieren mehrerer Plays .....	67
Angeleitete Übung: Implementieren mehrerer Plays .....	77
Praktische Übung: Implementieren von Playbooks .....	84
Zusammenfassung .....	91
<b>3. Verwalten von Variablen und Fakten</b>	<b>93</b>
Verwalten von Variablen .....	94
Angeleitete Übung: Verwalten von Variablen .....	103
Verwalten von Secrets .....	109
Angeleitete Übung: Verwalten von Secrets .....	115
Verwalten von Fakten .....	118
Angeleitete Übung: Verwalten von Fakten .....	128
Praktische Übung: Verwalten von Variablen und Fakten .....	133
Zusammenfassung .....	147
<b>4. Implementieren der Aufgabensteuerung</b>	<b>149</b>
Schreiben von Loops und bedingten Aufgaben .....	150
Angeleitete Übung: Loops und Aufgaben mit Bedingungen schreiben .....	161
Implementieren von Handlern .....	165
Angeleitete Übung: Implementieren von Handlern .....	168
Task-Fehler behandeln .....	173
Angeleitete Übung: Task-Fehler behandeln .....	178
Praktische Übung: Implementieren der Aufgabensteuerung .....	186
Zusammenfassung .....	196
<b>5. Bereitstellen von Dateien auf verwalteten Hosts</b>	<b>197</b>
Ändern und Kopieren von Dateien auf Hosts .....	198
Angeleitete Übung: Ändern und Kopieren von Dateien auf Hosts .....	205
Bereitstellen benutzerdefinierter Dateien mit Jinja2-Vorlagen .....	214
Angeleitete Übung: Bereitstellen benutzerdefinierter Dateien mit Jinja2-Vorlagen .....	220
Praktische Übung: Bereitstellen von Dateien auf verwalteten Hosts .....	223
Zusammenfassung .....	230
<b>6. Verwalten komplexer Plays und Playbooks</b>	<b>231</b>
Auswählen von Hosts mit Host-Pattern .....	232

Angeleitete Übung: Auswählen von Hosts mit Host-Pattern .....	241
Einbeziehen und Importieren von Dateien .....	248
Angeleitete Übung: Einbeziehen und Importieren von Dateien .....	254
Praktische Übung: Verwalten komplexer Plays und Playbooks .....	259
Zusammenfassung .....	267
<b>7. Playbooks mit Rollen vereinfachen</b>	<b>269</b>
Erläutern der Rollenstruktur .....	270
Quiz: Erläutern der Rollenstruktur .....	276
Wiederverwenden von Inhalten mit Systemrollen .....	278
Angeleitete Übung: Wiederverwenden von Inhalten mit Systemrollen .....	286
Erstellen von Rollen .....	292
Angeleitete Übung: Erstellen von Rollen .....	298
Bereitstellen von Rollen mit Ansible Galaxy .....	304
Angeleitete Übung: Bereitstellen von Rollen mit Ansible Galaxy .....	312
Abrufen von Rollen und Modulen aus Content-Sammlungen .....	320
Angeleitete Übung: Abrufen von Rollen und Modulen aus Content-Sammlungen .....	328
Praktische Übung: Vereinfachen von Playbooks mit Rollen .....	333
Zusammenfassung .....	345
<b>8. Fehlerbehebung in Ansible</b>	<b>347</b>
Fehlerbehebung in Playbooks .....	348
Angeleitete Übung: Fehlerbehebung in Playbooks .....	351
Fehlerbehebung auf verwalteten Ansible-Hosts .....	359
Angeleitete Übung: Fehlerbehebung auf verwalteten Ansible-Hosts .....	364
Praktische Übung: Fehlerbehebung in Ansible .....	368
Zusammenfassung .....	377
<b>9. Automatisieren von Linux-Administrationsaufgaben</b>	<b>379</b>
Verwalten von Software und Subskriptionen .....	380
Angeleitete Übung: Verwalten von Software und Subskriptionen .....	389
Verwalten von Benutzern und der Authentifizierung .....	397
Angeleitete Übung: Verwalten von Benutzern und der Authentifizierung .....	401
Verwalten des Bootvorgangs und geplanter Vorgänge .....	408
Angeleitete Übung: Verwalten des Boot-Prozesses und geplanter Prozesse .....	412
Verwalten von Storage .....	421
Angeleitete Übung: Verwalten von Storage .....	430
Verwalten der Netzwerkkonfiguration .....	443
Angeleitete Übung: Verwalten der Netzwerkkonfiguration .....	451
Praktische Übung: Automatisieren von Linux-Administrationsaufgaben .....	455
Zusammenfassung .....	469
<b>10. Ausführliche Wiederholung: Linux Automation with Ansible</b>	<b>471</b>
Ausführliche Wiederholung .....	472
Praktische Übung: Bereitstellen von Ansible .....	475
Praktische Übung: Erstellen von Playbooks .....	480
Praktische Übung: Erstellen von Rollen .....	487
<b>A. Ergänzende Themen</b>	<b>499</b>
Ermitteln möglicher Ansible-Konfigurationsoptionen .....	500
<b>B. Ansible Lightbulb-Lizenziierung</b>	<b>503</b>
Ansible Lightbulb-Lizenz .....	504

# Dokumentkonventionen

---

In diesem Abschnitt werden verschiedene Konventionen und Praktiken beschrieben, die in allen Red Hat Training-Kursen verwendet werden.

## Verweise

Die Red Hat Training-Kurse verwenden folgende Verweisarten:



### Literaturhinweise

Diese geben an, wo Sie weitere Informationen zu einem Thema in externen Dokumentationen finden können.



### Anmerkung

Diese sind Tipps, Tastenkombinationen oder alternative Ansätze für die vorliegende Aufgabe. Wenn Sie einen Hinweis ignorieren, hat dies normalerweise keine negativen Konsequenzen. Allerdings können Hinweise helfen, einen Vorgang zu optimieren.



### Wichtig

In diesen Feldern werden Details hervorgehoben, die andernfalls leicht übersehen werden könnten: Konfigurationsänderungen, die nur die aktuelle Sitzung betreffen, oder Services, die neu gestartet werden müssen, bevor ein Update angewendet werden kann. Wenn Sie diese ignorieren, führt dies nicht zu Datenverlust, kann jedoch Irritationen und Frustration hervorrufen.



### Warnung

Diese sollten nicht ignoriert werden. Wenn Sie diese ignorieren, führt dies mit großer Wahrscheinlichkeit zu einem Datenverlust.

## Inklusive Sprache

Red Hat Training überprüft derzeit die Verwendung der Sprache in verschiedenen Bereichen, um potenziell anstößige Begriffe zu entfernen. Dies ist ein fortlaufender Prozess und erfordert die Anpassung an die Produkte und Services, die in Red Hat Training-Kursen behandelt werden. Red Hat schätzt Ihre Geduld während dieses Prozesses.



# Einführung

## Red Hat Enterprise Linux-Automatisierung mit Ansible

Automatisierung von Red Hat Enterprise Linux mit Ansible (RH294) richtet sich an Linux-Systemadministratoren und -Entwickler, welche die Bereitstellung, Konfiguration, Anwendungsbereitstellung und Orchestrierung automatisieren möchten.

Die Kursteilnehmer lernen, Ansible auf einer Management-Workstation zu installieren und zu konfigurieren und verwaltete Hosts für die Automatisierung vorzubereiten. Die Kursteilnehmer schreiben Ansible-Playbooks, um Aufgaben (Tasks) zu automatisieren, und sie führen diese aus, um sicherzustellen, dass Server korrekt bereitgestellt und konfiguriert werden. Beispiele für Ansätze zur Automatisierung gängiger Linux-Systemverwaltungsaufgaben werden untersucht.

### Lerninhalte

- Installieren und Konfigurieren von Ansible über Red Hat Ansible Automation Platform auf einem Kontrollknoten
- Erstellen und Verwalten von Inventaren verwalteter Hosts sowie deren Vorbereitung für die Ansible-Automatisierung
- Ausführen einzelner Ad-hoc-Automatisierungsaufgaben über die Befehlszeile
- Schreiben von Ansible-Playbooks, um mehrere Aufgaben konsistent zu automatisieren und auf verwalteten Hosts anzuwenden
- Parametrisieren von Playbooks mithilfe von Variablen und Fakten und Schützen vertraulicher Daten mit Ansible Vault
- Schreiben und Wiederverwenden vorhandener Ansible-Rollen, um die Erstellung von Playbooks und die Wiederverwendung von Code zu vereinfachen
- Automatisieren gängiger Red Hat Enterprise Linux-Systemadministrationsaufgaben mit Ansible

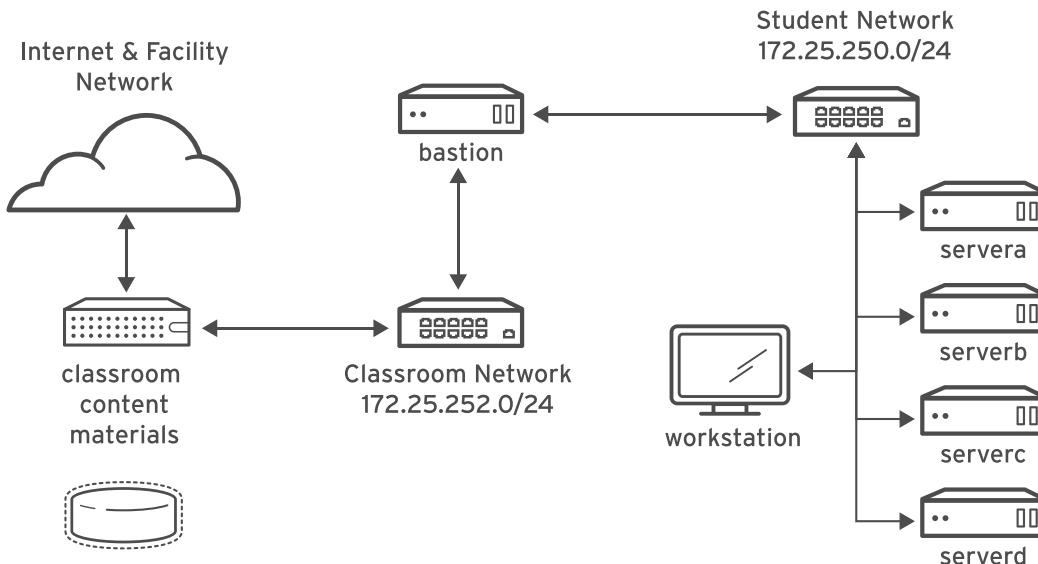
## Zielgruppe

- Linux-Systemadministratoren, DevOps-Ingenieure, Ingenieure für Infrastrukturautomatisierung und Systementwickler, die für die Automatisierung des Konfigurationsmanagements, die konsistente und wiederholbare Anwendungsbereitstellung, das Provisioning und die Bereitstellung von Entwicklungs-, Test- und Produktionsservern sowie für die Integration in DevOps-CI/CD-Arbeitsabläufe verantwortlich sind.

## Voraussetzungen

- Zertifizierung als Red Hat Certified System Administrator (EX200/RHCSA) oder gleichwertige Kenntnisse und Erfahrung in Bezug auf Red Hat Enterprise Linux.

# Informationen zur Kursumgebung



**Abbildung 0.1: Kursumgebung**

In diesem Kurs wird **workstation** als primäres Computersystem für praktische Übungen verwendet. Für diese Übungen werden außerdem vier weitere Rechner eingesetzt: **servera**, **serverb**, **serverc** und **serverd**. Alle fünf Systeme befinden sich in der DNS-Domäne `lab.example.com`.

Alle Kursteilnehmer-Systeme verfügen über das standardmäßige Benutzerkonto **student** mit dem Passwort **student**. Das **root**-Passwort für alle Kursteilnehmer-Systeme lautet **redhat**.

## Kursraum-Rechner

Rechnername	IP-Adressen	Rolle
bastion.lab.example.com	172.25.250.254	Gateway-System zum Verbinden des privaten Netzwerks des Teilnehmers mit dem Kursraumservern (muss immer ausgeführt werden)
workstation.lab.example.com	172.25.250.9	Grafische Workstation für die Systemadministration
servera.lab.example.com	172.25.250.10	Mit Ansible verwalteter Host
serverb.lab.example.com	172.25.250.11	Mit Ansible verwalteter Host
serverc.lab.example.com	172.25.250.12	Mit Ansible verwalteter Host

Rechnername	IP-Adressen	Rolle
serverd.lab.example.com	172.25.250.13	Mit Ansible verwalteter Host

Die primäre Funktion von **bastion** ist, dass es als Router zwischen dem Kursraumnetzwerk und dem Netzwerk fungiert, das die Rechner der Kursteilnehmer verbindet. Wenn **bastion** außer Betrieb ist, können andere Kursteilnehmer-Rechner nur auf Systeme im individuellen Kursteilnehmer-Netzwerk zugreifen.

Im Kursraum bieten verschiedene Systeme Unterstützung. Die beiden Server **content.example.com** und **materials.example.com** dienen als Quelle für Software- und Übungsmaterialien für praktische Übungen. Informationen zur Verwendung dieser Server finden Sie in der Anleitung der jeweiligen Übungen. Diese werden vom virtuellen Rechner **classroom.example.com** bereitgestellt. **classroom** und **bastion** sollten immer ausgeführt werden, damit die Übungsumgebung ordnungsgemäß verwendet wird.

## Steuerung Ihrer Systeme

### rht-vmctl-Befehle

Aktion	Befehl
server-Rechner starten	<code>rht-vmctl start server</code>
„Physische Konsole“ zum Anmelden anzeigen und mit Rechner Server arbeiten	<code>rht-vmview view server</code>
Server-Rechner auf seinen vorherigen Status zurücksetzen und virtuellen Rechner neu starten	<code>rht-vmctl reset server</code>

Kursteilnehmern werden Remote-Computer in einem Red Hat Online Learning-Kursraum zugewiesen. Der Zugriff darauf erfolgt über eine unter [rol.redhat.com](http://rol.redhat.com) [<http://rol.redhat.com>] gehostete Webanwendung. Kursteilnehmer sollten sich mithilfe ihrer Anmelddaten für das Red Hat Customer Portal auf dieser Website anmelden.

## Steuern der virtuellen Rechner

Die virtuellen Rechner in Ihrer Kursumgebung werden über eine Webseite gesteuert. Der Status jedes virtuellen Rechners im Kursraum wird auf der unter der Registerkarte **Online Lab** befindlichen Seite angezeigt.

### Rechnerstatus

VM-Status	Beschreibung
STARTING	Der virtuelle Rechner wird hochgefahren.
STARTED	Der virtuelle Rechner wird ausgeführt und ist verfügbar (oder, falls noch hochgefahren wird, wird es bald sein.)
STOPPING	Der virtuelle Rechner wird heruntergefahren.

VM-Status	Beschreibung
STOPPED	Der virtuelle Rechner ist vollständig heruntergefahren. Beim Starten fährt der virtuelle Rechner in denselben Status hoch, in dem er sich vor dem Herunterfahren befand. (Die Disk wurde nicht gelöscht.)
PUBLISHING	Der virtuelle Rechner wird anfänglich erstellt.
WAITING_TO_START	Der virtuelle Rechner wartet auf den Start anderer virtueller Rechner.

In Abhängigkeit des Status eines Rechners steht eine Auswahl der folgenden Aktionen zur Verfügung.

#### Aktionen für Kursumgebung/Rechner

Schaltfläche oder Aktion	Beschreibung
PROVISION LAB	Erstellen Sie den ROL-Kursraum. Hiermit werden sämtliche für die Kursumgebung erforderlichen virtuellen Rechner erstellt und gestartet. Dies dauert ggf. mehrere Minuten.
DELETE LAB	Entfernen Sie den ROL-Kursraum. Hiermit werden alle virtuellen Rechner im Kursraum entfernt. Achtung: Sämtliche auf den Disks gespeicherte Arbeit geht verloren.
START LAB	Starten Sie alle virtuellen Rechner im Kursraum.
SHUTDOWN LAB	Halten Sie alle virtuellen Rechner im Kursraum an.
OPEN CONSOLE	Öffnet eine neue Registerkarte im Browser und stellt eine Verbindung zwischen Konsole und virtuellem Rechner her. Kursteilnehmer können sich direkt beim virtuellen Rechner anmelden und Befehle ausführen. In den meisten Fällen sollten sich die Kursteilnehmer beim virtuellen Rechner workstation anmelden und ssh verwenden, um mit anderen virtuellen Rechnern eine Verbindung herzustellen.
ACTION → Start	Startet den virtuellen Rechner (d. h. schaltet ihn ein).
ACTION → Shutdown	Fährt den virtuellen Rechner ordnungsgemäß herunter, damit die Disk-Inhalte nicht verloren gehen.
ACTION → Power Off	Erzwingt ein Herunterfahren des virtuellen Rechners und behält die Inhalte seiner Disk bei. Dies entspricht der Stromabschaltung bei einem physischen Rechner.
ACTION → Reset	Erzwingen Sie das Herunterfahren des virtuellen Rechners, und setzen Sie die Disk in den Ursprungszustand zurück. Achtung: Sämtliche auf der Disk gespeicherte Arbeit geht verloren.

Klicken Sie zu Beginn einer Übung, sofern Sie angewiesen wurden, einen einzelnen Knoten eines virtuellen Rechners zurückzusetzen, nur für den bestimmten virtuellen Rechner auf ACTION → Reset.

## Einführung

Klicken Sie zu Beginn einer Übung, sofern Sie angewiesen wurden, alle virtuellen Rechner zurückzusetzen, auf ACTION → **Reset**.

Wenn Sie die Kursumgebung auf ihren ursprünglichen Zustand beim Start des Kurses zurücksetzen möchten, können Sie auf **DELETE LAB** klicken, um die gesamte Kursumgebung zu entfernen.

Nach dem Löschen des Labs können Sie auf **PROVISION LAB** klicken, um einen neuen Satz von Kurssystemen bereitzustellen.



### Warnung

Der Vorgang **DELETE LAB** kann nicht rückgängig gemacht werden. Die von Ihnen bis zu diesem Zeitpunkt in der Kursumgebung vorgenommene Arbeit geht verloren.

## Der Autostop-Timer

Die Registrierung bei Red Hat Online Learning ermöglicht Kursteilnehmern eine bestimmte Menge Zeit am Computer. Für den sparsamen Umgang mit der vorgegebenen Computerzeit verfügt der ROL-Kursraum über einen verknüpften Zählvorgang, der die Kursumgebung herunterfährt, wenn der Timer abgelaufen ist.

Klicken Sie zum Anpassen des Timers auf **MODIFY**, damit das Dialogfeld **New Autostop Time** angezeigt wird. Legen Sie die Anzahl der Stunden und Minuten fest, bis der Kursraum automatisch angehalten wird.

Klicken Sie auf **ADJUST TIME**, um diese Änderung auf die Timer-Einstellungen anzuwenden.

## Kapitel 1

# Einführung in Ansible

### Ziel

Beschreiben der grundlegenden Konzepte und der Verwendung von Ansible und Installieren von Ansible über Red Hat Ansible Automation Platform.

### Ziele

- Motivation für die Automatisierung von Linux-Administrationsaufgaben mit Ansible, grundlegende Ansible-Konzepte und grundlegende Architektur von Ansible beschreiben.
- Installieren von Ansible auf einem Kontrollknoten und Beschreiben der Unterschiede zwischen Community Ansible und Red Hat Ansible Automation Platform.

### Abschnitte

- Automatisieren der Linux-Administration mit Ansible (und Test)
- Installieren von Ansible (und angeleitete Übung)

# Automatisieren der Linux-Administration mit Ansible

---

## Ziel

Am Ende dieses Abschnitts sollten Sie zu Folgendem in der Lage sein: Motivation für die Automatisierung von Linux-Administrationsaufgaben mit Ansible, grundlegende Ansible-Konzepte und die grundlegende Architektur von Ansible beschreiben.

## Automatisierung und Linux-Systemadministration

Traditionell werden die meisten Aufgaben bei der Systemadministration und der Infrastrukturverwaltung manuell in einer grafischen Benutzeroberfläche oder in einer auf Befehlszeilen basierenden Oberfläche ausgeführt. Systemadministratoren arbeiten häufig mit Checklisten, sonstiger Dokumentation oder anhand von auswendig gelernten Routineverfahren, um Standardaufgaben auszuführen.

Dieser Ansatz ist jedoch fehleranfällig. Ein Systemadministrator kann dabei leicht einen Schritt überspringen oder einen Schritt versehentlich ausführen.

Häufig wird nur eingeschränkt verifiziert, ob die Schritte ordnungsgemäß ausgeführt wurden oder zu dem erwarteten Ergebnis führten.

Wenn jeder Server manuell und einzeln verwaltet wird, kann es darüber hinaus bei Servern, die eigentlich identisch konfiguriert sein sollten, vorkommen, dass die Konfiguration sich minimal (oder auch weitreichender) unterscheidet. Dies kann die Wartung erschweren und zu Fehlern oder einer instabilen IT-Umgebung führen.

Die *Automatisierung* ermöglicht es, Probleme zu vermeiden, die durch manuelle Systemadministration und Infrastrukturverwaltung verursacht werden. Als Systemadministrator können Sie Automatisierung verwenden, um sicherzustellen, dass alle Systeme schnell und ordnungsgemäß bereitgestellt und konfiguriert werden. Sie können so die sich wiederholenden Aufgaben im Tagesablauf automatisieren und damit Zeit sparen und sich auf wichtigere Dinge konzentrieren. Für Ihr Unternehmen bedeutet dies, dass Sie die nächste Version einer Anwendung oder Updates für einen Dienst schneller bereitstellen können.

## „Infrastructure as Code“

Ein gutes Automatisierungssystem ermöglicht Ihnen die Implementierung von *Infrastructure as Code*-Verfahren. „*Infrastructure as Code*“ bedeutet, dass Sie eine maschinenlesbare Automatisierungssprache verwenden können, um den Zustand zu definieren und zu beschreiben, in dem sich die IT-Infrastruktur befinden soll. Im Idealfall sollte diese Automatisierungssprache auch für den Menschen sehr einfach zu lesen sein, da Sie dann den Status leicht verstehen und Änderungen daran vornehmen können. Dieser Code wird dann auf die Infrastruktur angewendet, um sicherzustellen, dass sie sich tatsächlich in diesem Zustand befindet.

Wenn die Automatisierungssprache als einfache Textdateien vorliegen, kann sie problemlos in einem Versionskontrollsystem verwaltet werden. Dies hat den Vorteil, dass jede Änderung in das Versionskontrollsystem eingecheckt wird und Sie die Änderungen, die Sie im Laufe der Zeit vorgenommen haben, nachverfolgen können. Wenn Sie zu einer früheren Konfiguration zurückkehren möchten, von der Sie wissen, dass sie ordnungsgemäß funktioniert, können Sie diese Version einfach auschecken und auf die Infrastruktur anwenden.

Dies bildet die Grundlage, die Sie bei der Anwendung von bewährten Methoden in DevOps unterstützen. Entwickler können die gewünschte Konfiguration in der Automatisierungssprache definieren. Bediener können diese Änderungen einfacher überprüfen, um Feedback zu geben. Außerdem können sie diese Automatisierung nutzen, um reproduzierbar sicherzustellen, dass sich die Systeme in dem von den Entwicklern erwarteten Zustand befinden.

## Reduzierung menschlicher Fehler

Indem Sie die auf den Servern manuell ausgeführten Aufgaben durch Automatisierung und die Anwendung von Infrastructure as Code-Verfahren reduzieren, weisen diese Server häufiger eine konsistente Konfiguration auf. Dies bedeutet jedoch auch, dass Sie sich daran gewöhnen müssen, Änderungen durch Aktualisierung des Automatisierungscodes vorzunehmen, anstatt sie manuell auf die Server anzuwenden. Andernfalls laufen Sie Gefahr, dass manuell angewendete Änderungen verloren gehen, wenn Sie das nächste Mal Änderungen über die Automatisierung anwenden.

Die Automatisierung ermöglicht Ihnen, die Code-Prüfung, die Peer-Überprüfung durch mehrere Experten auf dem Fachgebiet und die Dokumentation des Verfahrens durch die Automatisierung selbst zu nutzen, um die Risiken im Betriebsablauf zu minimieren.

Letztendlich können Sie auch erzwingen, dass Änderungen an der IT-Infrastruktur nur durch Automatisierung vorgenommen werden können, um menschliche Fehler auf diese Weise zu minimieren.

## Was ist Ansible?

Ansible ist eine Open Source-Automatisierungsplattform. Hierbei handelt es sich um eine *einfache Automatisierungssprache*, die eine IT-Anwendungsstruktur in Ansible Playbooks perfekt beschreiben kann. Sie ist auch eine *Automatisierungs-Engine*, die Ansible-Playbooks ausführt.

Ansible kann leistungsfähige Automatisierungsaufgaben verwalten und lässt sich an viele verschiedene Workflows und Umgebungen anpassen. Gleichzeitig kann Ansible von neuen Benutzern schnell genutzt werden, um produktiver zu werden.

## Ansible ist einfach

Ansible-Playbooks ermöglichen eine für Benutzer lesbare Automatisierung. Dies bedeutet, dass Playbooks Automatisierungstools sind, die auch von Benutzern leicht gelesen, verstanden und geändert werden können. Zu ihrer Erstellung sind keine besonderen Programmierfähigkeiten erforderlich. Mit Playbooks werden Aufgaben nacheinander ausgeführt. Dank der Einfachheit des Playbook-Designs können sie von jedem Team verwendet werden, sodass unerfahrene Ansible-Benutzer schnell produktiv werden können.

## Ansible ist leistungsfähig

Ansible kann zur Bereitstellung von Anwendungen zur Konfigurationsverwaltung, zur Workflow-Automatisierung und zur Netzwerkautomatisierung eingesetzt werden. Ansible kann zur Orchestrierung des gesamten Lebenszyklus von Anwendungen genutzt werden.

## Ansible arbeitet ohne Agenten

Ansible basiert auf einer Architektur *ohne Agenten*. Normalerweise verwendet Ansible OpenSSH oder WinRM, um eine Verbindung mit den verwalteten Hosts herzustellen, und führt Aufgaben aus, indem oftmals (jedoch nicht immer) kleine Programme, die als *Ansible-Module* bezeichnet werden, an diese Hosts übertragen werden. Mithilfe dieser Programme wird das System in einen bestimmten gewünschten Status versetzt. Alle übertragenen Module werden entfernt, wenn

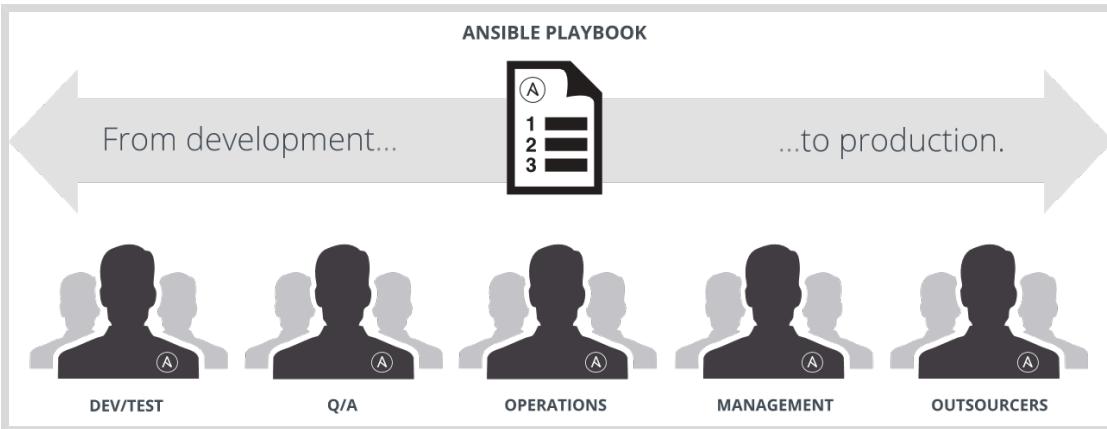
## Kapitel 1 | Einführung in Ansible

Ansible seine Aufgaben beendet hat. Sie können fast sofort mit der Verwendung von Ansible beginnen, weil keine speziellen Agenten für die Verwendung genehmigt und danach auf den verwalteten Hosts bereitgestellt werden müssen. Da es keine Agenten und keine zusätzliche benutzerdefinierte Sicherheitsinfrastruktur gibt, ist Ansible effizienter und sicherer als andere Alternativen.

Ansible besitzt eine Reihe wichtiger Stärken:

- *Plattformübergreifende Unterstützung*: Ansible bietet ohne Agenten Unterstützung für Linux-, Windows-, UNIX- und Netzwerkgeräte in physischen, virtuellen, Cloud- und Container-Umgebungen.
- *Für Benutzer lesbare Automatisierung*: Ansible-Playbooks, die als YAML-Textdateien erstellt werden, sind leicht lesbar und hilfreich, um sicherzustellen, dass jeder ihre Funktion versteht.
- *Perfekte Beschreibung von Anwendungen*: Jede Änderung kann mit Ansible-Playbooks vorgenommen werden und alle Aspekte Ihrer Anwendungsumgebung können beschrieben und dokumentiert werden.
- *Problemlose Verwaltung in der Versionskontrolle*: Ansible-Playbooks und -Projekte liegen im Klartext vor. Sie können wie Quellcode behandelt und in ihrem vorhandenen Versionskontrollsystem platziert werden.
- *Unterstützung für dynamische Inventare*: Die Liste der Rechner, die von Ansible verwaltet werden, kann dynamisch aus externen Quellen aktualisiert werden, um jederzeit und unabhängig von Infrastruktur und Standort die korrekte, aktuelle Liste aller verwalteten Server zu erfassen.
- *Problemlose Integration der Orchestrierung in andere Systeme*: HP SA, Puppet, Jenkins, Red Hat Satellite und andere in Ihrer Umgebung vorhandene Systeme können genutzt und in ihren Ansible-Workflow integriert werden.

## Ansible: Die Sprache von DevOps



**Abbildung 1.1: Ansible im gesamten Anwendungslebenszyklus**

Kommunikation ist der Schlüssel zu DevOps. Ansible ist die erste Automatisierungssprache, die überall in der IT gelesen und geschrieben werden kann. Ansible ist außerdem die einzige Automatisierungs-Engine, die den Anwendungslebenszyklus und die Pipeline für kontinuierliche Bereitstellung von Anfang bis Ende automatisieren kann.

## Konzepte und Architektur von Ansible

In der Ansible-Architektur gibt es zwei Arten von Rechnern: *Kontrollknoten* und *verwaltete Hosts*. Ansible wird auf einem Kontrollknoten installiert und ausgeführt, und dieser Rechner enthält auch

Kopien Ihrer Ansible-Projektdateien. Bei einem Kontrollknoten kann es sich um den Laptop eines Administrators, um ein von einer Reihe von Administratoren gemeinsam genutztes System oder um einen Server handeln, auf dem Red Hat Ansible Tower ausgeführt wird.

Verwaltete Hosts werden in einem *Inventar* aufgelistet, in dem diese Systeme außerdem zur einfacheren kollektiven Verwaltung in Gruppen organisiert werden. Das Inventar kann in einer statischen Textdatei definiert oder dynamisch mithilfe von Skripten ermittelt werden, die Informationen aus externen Quellen abrufen.

Anstatt komplexe Skripte zu schreiben, erstellen Ansible-Benutzer *Plays* auf hoher Ebene, um sicherzustellen, dass sich ein Host oder eine Hostgruppe in einem bestimmten Zustand befindet. Ein Play führt eine Reihe von *Aufgaben* auf den Hosts in der vom Play angegebenen Reihenfolge aus. Diese Plays werden im YAML-Format in einer Textdatei angegeben. Eine Datei, die ein oder mehrere Plays enthält, wird als ein *Playbook* bezeichnet.

Jede Aufgabe führt ein *Modul* aus, ein kurzer Codeabschnitt (geschrieben in Python, PowerShell oder einer anderen Sprache) mit bestimmten Argumenten. Jedes Modul ist im Grunde ein Tool in Ihrem Toolkit. Im Lieferumfang von Ansible sind mehrere Hundert nützlicher Module enthalten, die ein breites Spektrum von Automatisierungsaufgaben ausführen können. Sie können auf Systemdateien agieren, Software installieren oder API-Aufrufe vornehmen.

Wenn ein Modul in einer Aufgabe verwendet wird, stellt es im Allgemeinen sicher, dass ein bestimmter Aspekt im Zusammenhang mit dem Rechner einen bestimmten Status aufweist. Eine Aufgabe mit einem Modul kann z. B. sicherstellen, dass eine Datei vorhanden ist und über bestimmte Berechtigungen und Inhalte verfügt, während eine Aufgabe mit einem anderen Modul vielleicht gewährleistet, dass ein bestimmtes Dateisystem gemountet ist. Wenn das System nicht diesen Status aufweist, sollte es von der Aufgabe auf diesen Status gesetzt werden. Wenn das System bereits diesen Status aufweist, unternimmt die Aufgabe nichts. Wenn eine Aufgabe fehlschlägt, besteht das Standardverhalten von Ansible darin, das restliche Playbook für die Hosts abzubrechen, auf denen die Aufgabe fehlgeschlagen ist.

Aufgaben, Plays und Playbooks sind so konzipiert, dass sie *idempotent* sind. Folglich können Sie ein Playbook mehrfach ohne Weiteres auf denselben Hosts ausführen. Wenn sich Ihre Systeme im richtigen Zustand befinden, nimmt das Playbook keine Änderungen vor, wenn Sie es ausführen. Folglich sollten Sie ein Playbook mehrfach ohne Weiteres auf denselben Hosts ausführen können. Wenn sich Ihre Systeme im richtigen Zustand befinden, sollte das Playbook keine Änderungen vornehmen, wenn Sie es ausführen. Es gibt eine Handvoll Module, mit denen Sie beliebige Befehle ausführen können. Sie müssen diese Module jedoch mit Sorgfalt verwenden, um sicherzustellen, dass sie auf idempotente Weise ausgeführt werden.

Ansible verwendet außerdem *Plugins*. Bei Plugins handelt es sich um Code, den Sie Ansible hinzufügen können, um Ansible zu erweitern und an neue Anwendungsbereiche und Plattformen anzupassen.

Die Ansible-Architektur arbeitet ohne Agenten. Wenn ein Administrator ein Ansible-Playbook oder einen Ad-hoc-Befehl ausführt, verwendet der Kontrollknoten normalerweise SSH (standardmäßig) oder WinRM, um eine Verbindung mit dem verwalteten Host herzustellen. Dies bedeutet, dass Clients keinen Ansible-spezifischen Agenten benötigen, der auf verwalteten Hosts installiert ist, und keinen speziellen Netzwerkverkehr zu Nicht-Standard-Ports zulassen müssen.

## Unterstützung für Ansible

Die Red Hat Ansible Automation Platform ist eine vollständig unterstützte Version von Ansible, mit der Unternehmen ihre Automatisierung skalierbar verwalten können.

Mit diesem Tool erhalten Sie:

## Kapitel 1 | Einführung in Ansible

- Offizielle Unterstützung für das Kern-Toolset von Ansible
- Zertifizierte Content-Sammlungen, die Ihnen helfen, die Einführung von Ansible-Automatisierung mit unterstütztem Code zu beschleunigen
- Cloud-Services, mit denen Sie zertifizierte Ansible-Inhalte entdecken, die Zusammenarbeit im Team erleichtern und Betriebsanalysen zur Automatisierung gemischter, hybrider Umgebungen bereitstellen können
- On-Premise-Tools, mit denen Sie die Verwaltung von Automatisierungsaufgaben zentralisieren können

Die Automation Controller-Komponente (früher Red Hat Ansible Tower genannt) beispielsweise ist ein Enterprise-Framework, mit dem Sie unter anderem steuern können, wer Zugriff auf die Ausführung von Playbooks auf welchen Hosts besitzt. Sie können SSH-Anmelddaten gemeinsam nutzen, ohne Benutzern zu gestatten, sie zu übertragen oder ihren Inhalt anzuzeigen, Sie können Ihre gesamten Ansible-Jobs protokollieren und das Inventar verwalten.

Sie stellt eine browserbasierte Benutzeroberfläche (Webbenutzeroberfläche) und eine RESTful-API bereit. Die Ansible-Upstream-Community schließt diese Komponente nicht automatisch in das Ansible-Kernprodukt ein. Sie wird als Open Source entwickelt und als Teil des Produkts Red Hat Ansible Automation Platform bereitgestellt und unterstützt.

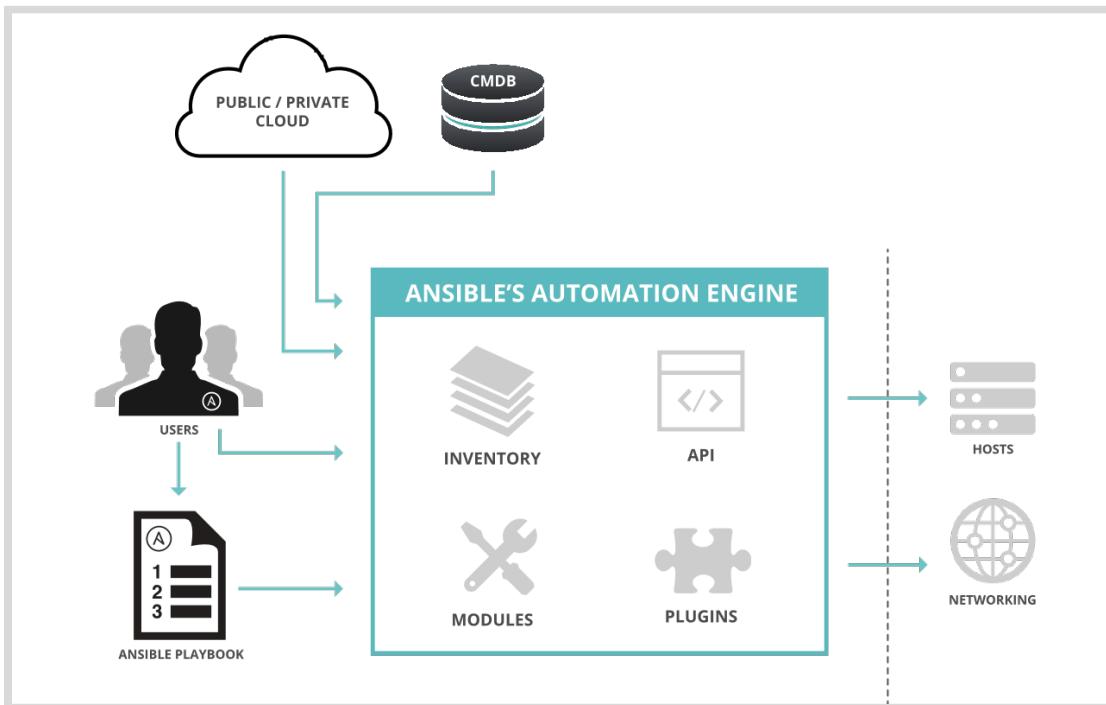


Abbildung 1.2: Ansible-Architektur

## Die Ansible-Methode

### Komplexität vernichtet Produktivität

Je einfacher, desto besser. Ansible ist so konzipiert, dass seine Tools einfach zu verwenden sind und die Automatisierung sich leicht schreiben und lesen lässt. Sie sollten dies nutzen, wenn Sie bei der Erstellung Ihrer Automatisierung nach Vereinfachung streben.

## Optimiert für Lesbarkeit

Die Automatisierungssprache Ansible basiert auf einfachen, selbst erklärenden und textbasierten Dateien, die für Benutzer leicht lesbar sind. Wenn Ansible-Playbooks ordnungsgemäß erstellt werden, können sie Ihre Workflow-Automatisierung übersichtlich dokumentieren.

## Selbst erklärendes Denken

Ansible ist eine *Engine für den gewünschten Status*. Sie begegnet dem Problem der Automatisierung von IT-Bereitstellungen, indem Sie sie in Form des Status ausdrücken, den Ihre Systeme aufweisen sollen. Das Ziel von Ansible besteht darin, Ihre Systeme auf den gewünschten Status zu setzen und nur die Änderungen vorzunehmen, die notwendig sind. Ansible wie eine Skripting-Sprache zu behandeln, ist nicht der richtige Ansatz.

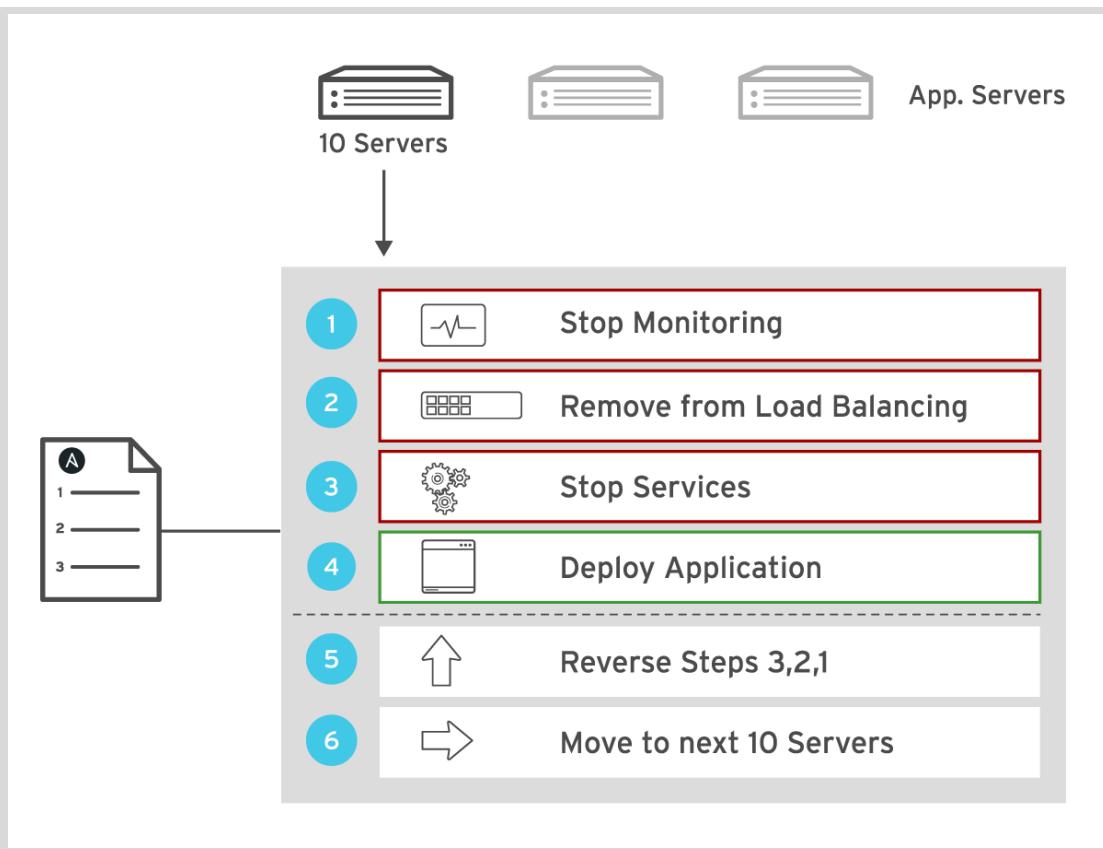


Abbildung 1.3: Ansible ermöglicht umfassende Automatisierung

## Anwendungsfälle

Im Gegensatz zu anderen Tools kombiniert und vereinigt Ansible die Orchestrierung mit Konfigurationsverwaltung, Provisioning und Anwendungsbereitstellung in einer benutzerfreundlichen Plattform.

Die Anwendungsfälle für Ansible umfassen unter anderem:

### Konfigurationsverwaltung

Die Zentralisierung der Verwaltung und Bereitstellung von Konfigurationsdateien ist ein häufig auftretender Anwendungsfall für Ansible, über den viele Power User zum ersten Mal mit der Ansible-Automatisierungsplattform in Kontakt kommen.

## Anwendungsbereitstellung

Wenn Sie Ihre Anwendung mit Ansible definieren und die Bereitstellung mit Red Hat Ansible Tower verwalten, können Teams den gesamten Lebenszyklus der Anwendung von der Entwicklung bis zur Produktion effizient verwalten.

## Provisioning

Anwendungen müssen auf Systemen bereitgestellt oder installiert werden. Ansible und Red Hat Ansible Tower können dazu beitragen, den Prozess der Bereitstellung von Systemen zu optimieren, unabhängig davon, ob Sie Bare-Metal-Server oder virtuelle Rechner per PXE-Booting und Kickstart booten oder mit Vorlagen virtuelle Rechner oder Cloud-Instanzen erstellen. Anwendungen müssen auf Systemen bereitgestellt oder installiert werden.

## Kontinuierliche Bereitstellung

Der Aufbau einer Pipeline für kontinuierliche Integration/Bereitstellung (CI/CD) erfordert die Koordinierung und Billigung zahlreicher Teams. Dieses Vorhaben kann ohne eine einfache Automatisierungsplattform, die jeder in Ihrer Organisation verwenden kann, nicht gelingen. Ansible-Playbooks sorgen dafür, dass Ihre Anwendungen während ihres gesamten Lebenszyklus ordnungsgemäß bereitgestellt und verwaltet werden.

## Sicherheit und Compliance

Wenn Ihre Sicherheitsrichtlinie in Ansible Playbooks definiert wird, können Scanning und Problembehebung übergreifender Sicherheitsrichtlinien problemlos in andere automatisierte Prozesse integriert werden.

So wird dieser Aspekt nicht zu einer Nebensache, sondern zu einem festen Bestandteil sämtlicher Bereitstellungen.

## Orchestrierung

Mit Konfigurationen allein lässt sich Ihre Umgebung nicht definieren. Sie müssen festlegen, wie verschiedene Konfigurationen interagieren, und sicherstellen, dass separate Elemente gemeinsam verwaltet werden können.



### Literaturhinweise

#### Ansible

<https://www.ansible.com>

#### Funktionsweise von Ansible

<https://www.ansible.com/how-ansible-works>

## ► Quiz

# Automatisieren der Linux-Administration mit Ansible

Wählen Sie die richtige Antwort auf die folgenden Fragen aus:

- ▶ 1. Welcher der folgenden Begriffe beschreibt am besten die Architektur von Ansible?
  - a. Ohne Agenten
  - b. Client/Server
  - c. Ereignisorientiert
  - d. Stateless
  
- ▶ 2. Welches Netzwerkprotokoll verwendet Ansible standardmäßig für die Kommunikation mit verwalteten Knoten?
  - a. HTTP
  - b. HTTPS
  - c. SNMP
  - d. SSH
  
- ▶ 3. In welcher der folgenden Dateien sind die Aktionen definiert, die Ansible auf verwalteten Knoten ausführt?
  - a. Hostinventar
  - b. Manifest
  - c. Playbook
  - d. Skript
  
- ▶ 4. In welcher Syntax werden Ansible-Playbooks definiert?
  - a. Bash
  - b. Perl
  - c. Python
  - d. YAML

## ► Lösung

# Automatisieren der Linux-Administration mit Ansible

Wählen Sie die richtige Antwort auf die folgenden Fragen aus:

► 1. Welcher der folgenden Begriffe beschreibt am besten die Architektur von Ansible?

- a. Ohne Agenten
- b. Client/Server
- c. Ereignisorientiert
- d. Stateless

► 2. Welches Netzwerkprotokoll verwendet Ansible standardmäßig für die Kommunikation mit verwalteten Knoten?

- a. HTTP
- b. HTTPS
- c. SNMP
- d. SSH

► 3. In welcher der folgenden Dateien sind die Aktionen definiert, die Ansible auf verwalteten Knoten ausführt?

- a. Hostinventar
- b. Manifest
- c. Playbook
- d. Skript

► 4. In welcher Syntax werden Ansible-Playbooks definiert?

- a. Bash
- b. Perl
- c. Python
- d. YAML

# Installieren von Ansible

## Ziele

Am Ende dieses Abschnitts sollten Sie in der Lage sein, Ansible auf einem Kontrollknoten zu installieren und die Unterschiede zwischen Community Ansible und Red Hat Ansible Automation Platform zu beschreiben.

## Ansible und Red Hat Ansible Automation Platform

Red Hat stellt eine vollständig unterstützte Version von Ansible über Red Hat Ansible Automation Platform bereit. Ansible Automation Platform bietet das Kern-Toolset von Ansible sowie zusätzliche zertifizierte und unterstützte Inhalte, Tools und Cloud-Services. Kunden mit einem gültigen Abonnement können das verfügbare Repository verwenden, die zusätzlichen Tools installieren und zertifizierte Inhalte aus den Cloud-Diensten nutzen.

Dieser Kurs basiert derzeit auf Red Hat Ansible Automation Platform 1.2, die Ansible 2.9 enthält.



### Anmerkung

Frühere Versionen von Red Hat Ansible Automation Platform beziehen sich auf die enthaltene Version von Ansible als „Red Hat Ansible Engine“, und Sie werden in einigen Dokumentationen auf diese Terminologie stoßen.

Die Upstream-Entwicklungs-Community bietet auch eine nicht unterstützte Version von Ansible. Diese wird bisher als RPM-Pakete bereitgestellt, soll aber bald ausschließlich über den Python Package Index (PyPI) bereitgestellt werden.

## Kontrollknoten

Ansible lässt sich leicht installieren. Die Ansible-Software muss lediglich auf dem oder den Kontrollknoten installiert werden, über die Ansible ausgeführt wird. Auf Hosts, die von Ansible verwaltet werden, muss Ansible nicht installiert sein.

Diese Installation des Kern-Toolsets von Ansible umfasst nur wenige Schritte und stellt minimale Anforderungen. Andererseits erfordert die Installation der zusätzlichen Komponenten, die Red Hat Ansible Automation Platform bereitstellt, wie z. B. des Automation Controller (früher Red Hat Ansible Tower genannt), Red Hat Enterprise Linux 8.2 oder höher mit mindestens zwei CPUs, 4 GiB RAM und 20 GiB verfügbarem Speicherplatz.

Auf dem Kontrollknoten muss Python 3 (Version 3.5 oder höher) oder Python 2 (Version 2.7 oder höher) installiert sein.



### Wichtig

Wenn Sie Red Hat Enterprise Linux 8 verwenden, kann Ansible automatisch das *platform-python*-Paket verwenden, das Systemdienstprogramme unterstützt, die Python nutzen. Sie müssen das *python36*- oder *python27*-Paket von AppStream nicht installieren.

```
[root@controlnode ~]# yum list installed platform-python
Installed Packages
platform-python.x86_64          3.6.8-37.el        @anaconda
```

Sie benötigen ein gültiges Abonnement für Red Hat Ansible Automation Platform, um das Kern-Toolset auf Ihrem Kontrollknoten zu installieren. Der Installationsprozess sieht folgendermaßen aus:

Wenn Sie Simple Content Access für Ihr Unternehmen im Red Hat Customer Portal aktiviert haben, müssen Sie das Abonnement nicht Ihrem System zuordnen. Der Installationsprozess sieht folgendermaßen aus:



### Warnung

Sie müssen diese Schritte nicht in Ihrer Kursumgebung ausführen.

- Registrieren Sie Ihr System bei Red Hat Subscription Manager.

```
[root@host ~]# subscription-manager register
```

- Aktivieren Sie das Red Hat Ansible Engine-Repository.

```
[root@host ~]# subscription-manager repos \
> --enable ansible-2-for-rhel-8-x86_64-rpms
```

- Installieren Sie Red Hat Ansible Engine.

```
[root@host ~]# yum install ansible
```

## Verwaltete Hosts

Einer der Vorteile von Ansible besteht darin, dass auf verwalteten Hosts kein spezieller Agent installiert werden muss. Der Ansible-Kontrollknoten stellt Verbindungen mit verwalteten Hosts mithilfe eines Standard-Netzwerkprotokolls her, um sicherzustellen, dass die Systeme den angegebenen Status aufweisen.

Verwaltete Hosts stellen möglicherweise einige Anforderungen, die davon abhängig sind, wie der Kontrollknoten eine Verbindung mit ihnen herstellt und welche Module auf ihm ausgeführt werden.

Damit auf verwalteten Linux- und UNIX-Hosts die meisten Module funktionieren, muss Python 2 (Version 2.6 oder höher) oder Python 3 (Version 3.5 oder höher) installiert sein.

Bei Red Hat Enterprise Linux 8 können Sie möglicherweise auch von *platform-python* abhängen. Sie können auch den *python36*-Anwendungsstream aktivieren und installieren (oder den *python27*-Anwendungsstream).

```
[root@host ~]# yum module install python36
```

Wenn auf den verwalteten Hosts SELinux aktiviert ist, müssen Sie sicherstellen, dass das Paket *python3-libselinux* installiert ist, bevor Sie Module verwenden, die sich auf Kopier-, Datei- oder Vorlagenfunktionen beziehen. (Hinweis: Wenn die anderen Python-Komponenten installiert sind, kann mithilfe von Ansible-Modulen wie *yum* oder *package* sichergestellt werden, dass dieses Paket ebenfalls installiert wird.)



### Wichtig

Einige Paketnamen können in Red Hat Enterprise Linux 7 und früheren Versionen anders lauten, was auf die fortlaufende Migration zu Python 3 zurückzuführen ist.

Installieren Sie für Red Hat Enterprise Linux 7 und ältere Versionen das *python*-Paket, das Python 2 bereitstellt. Installieren Sie statt *python3-libselinux* das *libselinux-python*-Paket.

Manche Module haben möglicherweise eigene zusätzliche Anforderungen. Das *dnf*-Modul etwa, das zur Installation von Paketen auf aktuellen Fedora-Systemen verwendet werden kann, benötigt das *python3-dnf*-Paket (*python-dnf* in RHEL 7).



### Anmerkung

Einige Module benötigen Python nicht. Argumente, die an das Ansible-Modul *raw* übergeben werden, werden z. B. direkt über die konfigurierte Remote-Shell ausgeführt, statt das Modulsubsystem zu durchlaufen. Dies kann nützlich sein, um Geräte zu verwalten, auf denen Python nicht verfügbar ist oder nicht installiert werden kann, oder um Python per Bootstrap auf ein System zu laden, auf dem es nicht enthalten ist.

Es ist jedoch schwierig, das Modul *raw* auf sichere, idempotente Weise zu verwenden. Wenn Sie stattdessen ein normales Modul verwenden können, ist es daher im Allgemeinen besser, den Einsatz von *raw* und ähnlichen command-Modulen zu vermeiden. Dies wird später in diesem Kurs behandelt.

## Verwaltete Hosts auf Basis von Microsoft Windows

Ansible umfasst eine Reihe von Modulen, die speziell für Microsoft Windows-Systeme konzipiert sind. Diese werden im Abschnitt Windows-Module [[https://docs.ansible.com/ansible/2.9/modules/list\\_of\\_windows\\_modules.html](https://docs.ansible.com/ansible/2.9/modules/list_of_windows_modules.html)] des Ansible-Modulindizes aufgelistet.

Für die meisten der speziell für verwaltete Microsoft Windows-Hosts konzipierten Module wird PowerShell 3.0 oder höher und nicht Python benötigt. Zusätzlich muss auf den verwalteten Hosts Windows PowerShell-Remoting konfiguriert sein. Für Ansible muss mindestens .NET Framework 4.0 oder höher auf verwalteten Windows-Hosts installiert werden.

Dieser Kurs verwendet in seinen Beispielen verwaltete Hosts auf Linux-Basis und befasst sich nicht sehr ausführlich mit den besonderen Unterschieden und Anpassungen, die bei der Verwaltung von Microsoft Windows-basierten verwalteten Hosts notwendig sind. Weitere Informationen finden Sie auf der Ansible-Website unter [https://docs.ansible.com/ansible/2.9/user\\_guide/windows.html](https://docs.ansible.com/ansible/2.9/user_guide/windows.html).

## Verwaltete Netzwerkgeräte

Sie können Ansible Automation auch verwenden, um verwaltete Netzwerkgeräte wie Router und Switches zu konfigurieren. Ansible umfasst eine Vielzahl von speziell für diesen Zweck entwickelten Modulen. Dies umfasst u. a. die Unterstützung für Cisco IOS, IOS XR und NX-OS, Juniper Junos, Arista EOS und VyOS-basierten Netzwerkgeräten.

Mithilfe derselben grundlegenden Techniken, die Sie beim Schreiben von Playbooks für Server verwenden, können Sie Ansible-Playbooks für Netzwerkgeräte schreiben. Da die meisten Netzwerkgeräte Python nicht ausführen können, führt Ansible Netzwerkmodule auf dem Kontrollknoten und nicht auf den verwalteten Hosts aus. Spezielle Verbindungsmethoden werden auch verwendet, um mit Netzwerkgeräten zu kommunizieren, normalerweise entweder CLI über SSH, XML über SSH oder API über HTTP(S).

In diesem Kurs wird die Automatisierung der Netzwerkgeräteverwaltung nicht behandelt. Weitere Informationen zu diesem Thema finden Sie unter *Ansible for Network Automation* (Ansible für die Netzwerkautomatisierung) [<https://docs.ansible.com/ansible/2.9/network/index.html>] auf der Ansible-Community-Website, oder besuchen Sie unseren Alternativkurs *Ansible for Network Automation* (DO457) (Ansible für die Netzwerkautomatisierung) [<https://www.redhat.com/en/services/training/do457-ansible-network-automation>].

## Vorbereitung auf Änderungen an den Freigabemethoden für Ansible

Sowohl die Ansible-Upstream-Community als auch die Red Hat Ansible Automation Platform durchlaufen Veränderungen hinsichtlich der Verpackung und Verteilung von Ansible an Benutzer.

Ansible 2.9, Ansible in Red Hat Ansible Automation Platform 1.2 und frühere Versionen von beiden wurden als RPM-Paket (`ansible`) bereitgestellt. Dieses Paket enthielt auch alle Ansible-Module und -Plugins.

In zukünftigen Versionen von Red Hat Ansible Automation Platform wird der Code, der die Automatisierung ausführt, in das neue Paket `ansible-core` verschoben, und unterstützte Module und Plugins werden mithilfe einer neuen Funktion, den *Content-Sammlungen*, bereitgestellt. Content-Sammlungen werden später im Kurs ausführlicher behandelt. Darüber hinaus umfasst Ansible Automation Platform 2 auch erweiterte Tools und Funktionen zum Ausführen Ihrer Playbooks, neue Cloud-Services-Funktionen und erweiterte Versionen des Automation Controller (früher Red Hat Ansible Tower genannt) und des Automation Hub.

In zukünftigen Versionen von Ansible aus der Community werden die ausführbaren Dateien und ein ausgewählter Satz von Inhalten über den Python Package Index (PyPI) bereitgestellt, von dem aus sie mit dem Befehl `pip install ansible` installiert werden können. Dieser ausgewählte Inhalt kann sich jedoch von dem unterscheiden, was Red Hat in der Red Hat Ansible Automation Platform unterstützt und zertifiziert.

Der Automatisierungscode, die Tools und die Techniken, die Sie in diesem Kurs lernen werden, gelten mit geringen oder keinen Änderungen direkt für zukünftige Versionen von Ansible.



### Literaturhinweise

Manpage ansible-doc(1)

**Knowledgebase: „How Do I Download and Install Red Hat Ansible Engine?“ (Wie kann Red Hat Ansible Engine heruntergeladen und installiert werden?)**

<https://access.redhat.com/articles/3174981>

**Simple Content Access**

<https://access.redhat.com/articles/simple-content-access>

**Produktdokumentation für Red Hat Ansible Automation Platform 1.2**

[https://access.redhat.com/documentation/en-us/red\\_hat\\_ansible\\_automation\\_platform/1.2/](https://access.redhat.com/documentation/en-us/red_hat_ansible_automation_platform/1.2/)

**Windows-Leitfäden – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/user\\_guide/windows.html](https://docs.ansible.com/ansible/2.9/user_guide/windows.html)

**Ansible for Networking Automation – Ansible-Dokumentation**

<https://docs.ansible.com/ansible/2.9/network/index.html>

## ► Angeleitete Übung

# Installieren von Ansible

In dieser Übung installieren Sie Ansible auf einem Kontrollknoten, auf dem Red Hat Enterprise Linux ausgeführt wird.

## Ergebnisse

Sie sollten in der Lage sein, Ansible auf einem Kontrollknoten zu installieren.

## Bevor Sie Beginnen

Melden Sie sich als Benutzer `student` mit dem Passwort `student` bei `workstation` an, und führen Sie `lab intro-install start` aus. Mit diesem Befehl wird der Kontrollknoten konfiguriert.

```
[student@workstation ~]$ lab intro-install start
```

## Anweisungen

- 1. Installieren Sie Ansible auf `workstation`, damit Sie diesen Rechner als Kontrollknoten verwenden können.

```
[student@workstation ~]$ sudo yum install ansible
[sudo] password for student: student
Last metadata expiration check: 0:00:44 ago on Thu 22 Jul 2021 01:27:41 AM EDT.
Dependencies resolved.
...output omitted...
Is this ok [y/d/N]: y
...output omitted...
```

- 2. Stellen Sie sicher, dass Ansible auf dem System installiert ist. Führen Sie den Befehl `ansible` mit der Option `--version` aus.

```
[student@workstation ~]$ ansible --version
ansible 2.9.15
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/student/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3.6/site-packages/ansible
  executable location = /usr/bin/ansible
  python version = 3.6.8 (default, Mar 18 2021, 08:58:41) [GCC 8.4.1 20200928 (Red
  Hat 8.4.1-1)]
```

- 3. Rufen Sie das Modul `setup` auf dem lokalen Host auf, um den Wert des Fakts `ansible_python_version` abzurufen.

```
[student@workstation ~]$ ansible -m setup localhost | grep ansible_python_version  
"ansible_python_version": "3.6.8",
```

## Beenden

Führen Sie auf `workstation` das Skript `lab intro-install finish` aus, um diese Übung zu bereinigen.

```
[student@workstation ~]$ lab intro-install finish
```

Hiermit ist die angeleitete Übung beendet.

# Zusammenfassung

---

In diesem Kapitel wurden die folgenden Themen behandelt:

- Die Automatisierung ist ein wichtiges Instrument, um menschliche Fehler zu minimieren und schnell sicherzustellen, dass sich Ihre IT-Infrastruktur in einem konsistenten und korrekten Zustand befindet.
- Ansible ist eine Open-Source-Automatisierungsplattform, die sich an viele verschiedene Workflows und Umgebungen anpassen lässt.
- Mithilfe von Ansible können viele verschiedene Arten von Systemen verwaltet werden, darunter Server unter Linux, Microsoft Windows oder UNIX sowie Netzwerkgeräte.
- Ansible-Playbooks sind vom Benutzer lesbare Textdateien, die den gewünschten Status einer IT-Infrastruktur beschreiben.
- Ansible basiert auf einer Architektur ohne Agenten, in der Ansible auf einem Kontrollknoten installiert wird und Clients keine spezielle Agentensoftware benötigen.
- Ansible stellt mithilfe von Standard-Netzwerkprotokollen wie SSH Verbindungen mit verwalteten Hosts her und führt Code oder Befehle auf den verwalteten Hosts aus, um sicherzustellen, dass sie den von Ansible festgelegten Status aufweisen.

## Kapitel 2

# Implementieren eines Ansible-Playbooks

### Ziel

Erstellen eines Inventars verwalteter Hosts, Schreiben eines einfachen Ansible-Playbooks und Ausführen des Playbooks, um Aufgaben auf diesen Hosts zu automatisieren.

### Ziele

- Beschreiben der Konzepte von Ansible-Inventaren und Verwalten einer statischen Inventar-Datei.
- Beschreiben, wo Ansible-Konfigurationsdateien gespeichert sind, wie Ansible diese auswählt, und diese bearbeiten, um Änderungen an Standardeinstellungen vorzunehmen.
- Eine einzelne Ansible-Automatisierungsaufgabe mit einem Ad-hoc-Befehl ausführen und einige Anwendungsfälle für Ad-hoc-Befehle erläutern.
- Ein einfaches Ansible-Playbook schreiben und dieses Playbook mit dem Befehl `ansible-playbook` ausführen.
- Schreiben eines Playbooks, das mehrere Plays und die Rechteerweiterung auf Play-Basis verwendet, und effektives Verwenden von `ansible-doc`, um zu lernen, wie Sie neue Module verwenden, um Aufgaben für ein Play zu implementieren.

### Abschnitte

- Erstellen eines Ansible-Inventars (und angeleitete Übung)
- Verwalten von Ansible-Konfigurationsdateien (und angeleitete Übung)
- Ausführen von Ad-hoc-Befehlen (und angeleitete Übung)
- Schreiben und Ausführen von Playbooks (und angeleitete Übung)
- Implementieren mehrerer Plays (und angeleitete Übung)

## Praktische Übung

- Implementieren von Playbooks

# Erstellen eines Ansible-Inventars

## Ziele

Nach Abschluss dieses Kapitels sollten Sie in der Lage sein, die Konzepte von Ansible-Inventaren zu beschreiben und eine statische Inventardatei zu verwalten.

## Definieren des Inventars

Ein *Inventar* definiert eine Sammlung von Hosts, die von Ansible verwaltet werden. Diese Hosts können auch *Gruppen* zugewiesen werden, die gemeinsam verwaltet werden können. Gruppen können untergeordnete Gruppen enthalten und Hosts können Mitglied mehrerer Gruppen sein. Mit dem Inventar können auch Variablen festgelegt werden, die auf die vom Inventar definierten Hosts und Gruppen angewendet werden.

Hostinventare können in zweierlei Weise definiert werden. Ein *statisches* Hostinventar kann durch eine Textdatei definiert werden. Ein *dynamisches* Hostinventar kann mithilfe externer Informationsanbieter je nach Bedarf von einem Skript oder einem anderen Programm generiert werden.

## Angeben verwalteter Hosts mit einem statischen Inventar

Eine statische Inventardatei ist eine Textdatei, in der die verwalteten Hosts angegeben werden, die Ansible anspricht. Sie können diese Datei unter Verwendung verschiedener Formate schreiben, einschließlich eines INI-Formats oder YAML. Das INI-Format ist sehr verbreitet und wird für die meisten Beispiele in diesem Kurs verwendet.



### Anmerkung

Es werden mehrere statische Bestandsformate von Ansible unterstützt. In diesem Abschnitt konzentrieren wir uns auf das gängigste, das INI-Format.

Die einfachste Form einer statischen Inventardatei im INI-Format ist eine Liste von Hostnamen oder IP-Adressen verwalteter Hosts in jeweils einer Zeile:

```
web1.example.com
web2.example.com
db1.example.com
db2.example.com
192.0.2.42
```

Normalerweise werden verwaltete Hosts jedoch in *Hostgruppen* organisiert. Hostgruppen ermöglichen es Ihnen, Ansible effektiver auf einer Sammlung von Systemen auszuführen. In diesem Fall beginnt jeder Abschnitt mit dem Namen einer Hostgruppe, der in eckige Klammern ([]) eingeschlossen ist. Danach folgt jeweils in einer Zeile der Hostname oder eine IP-Adresse für jeden verwalteten Host in der Gruppe.

## Kapitel 2 | Implementieren eines Ansible-Playbooks

Im folgenden Beispiel definiert das Hostinventar die zwei Hostgruppen `webservers` und `db-servers`.

```
[webservers]
web1.example.com
web2.example.com
192.0.2.42

[db-servers]
db1.example.com
db2.example.com
```

Hosts können in mehreren Gruppen enthalten sein. Es ist tatsächlich die empfohlene Vorgehensweise, Ihre Hosts in mehreren Gruppen und vielleicht je nach Rolle des Hosts, seinem physischen Standort, seinem Produktionsstatus usw. in unterschiedlicher Weise zu organisieren. So können Sie Ansible-Plays einfach auf bestimmte Hosts anwenden.

```
[webservers]
web1.example.com
web2.example.com
192.0.2.42

[db-servers]
db1.example.com
db2.example.com

[east-datacenter]
web1.example.com
db1.example.com

[west-datacenter]
web2.example.com
db2.example.com

[production]
web1.example.com
web2.example.com
db1.example.com
db2.example.com

[development]
192.0.2.42
```



### Wichtig

Zwei Hostgruppen sind immer vorhanden:

- Die Hostgruppe `all` enthält alle Hosts, die im Inventar explizit aufgelistet sind.
- Die Hostgruppe `ungrouped` enthält alle Hosts, die im Inventar explizit aufgelistet und kein Mitglied einer anderen Gruppe sind.

## Definieren verschachtelter Gruppen

Ansible-Hostinventare können Gruppen von Hostgruppen enthalten. Dies wird erreicht, indem eine Hostgruppe mit dem Namenssuffix :children erstellt wird. Das folgende Beispiel erstellt eine neue Gruppe mit dem Namen `north-america`, die alle Hosts aus den Gruppen `usa` und `canada` enthält.

```
[usa]
washington1.example.com
washington2.example.com

[canada]
ontario01.example.com
ontario02.example.com

[north-america:children]
canada
usa
```

Eine Gruppe kann sowohl verwaltete Hosts als auch untergeordnete Gruppen als Mitglieder enthalten. Im vorherigen Inventar könnten Sie z. B. einen Abschnitt `[north-america]` hinzufügen, der eine eigene Liste verwalteter Hosts enthält. Diese Liste von Hosts würde mit den zusätzlichen Hosts zusammengeführt, die die Gruppe `north-america` von ihren untergeordneten Gruppen geerbt hat.

## Vereinfachen von Hostspezifikationen mit Bereichen

Sie können Bereiche in den Hostnamen oder IP-Adressen angeben, um die Ansible-Hostinventare zu vereinfachen. Sie können entweder numerische oder alphabetische Bereiche angeben. Bereiche besitzen die folgende Syntax:

```
[START:END]
```

Bereiche entsprechen allen Werten von `START` bis einschließlich `END`. Sehen Sie sich die folgenden Beispiele an:

- „192.168.[4:7].[0:255]“ entspricht allen IPv4-Adressen im Netzwerk „192.168.4.0/22“ („192.168.4.0“ bis „192.168.7.255“).
- „server[01:20].example.com“ entspricht allen Hosts mit den Namen „server01.example.com“ bis „server20.example.com“.
- „[a:c].dns.example.com“ entspricht Hosts mit den Namen „a.dns.example.com“, „b.dns.example.com“ und „c.dns.example.com“.
- „2001:db8::[a:f]“ entspricht allen IPv6-Adressen von „2001:db8::a“ bis „2001:db8::f“.

Wenn führende Nullen in numerischen Bereichen vorhanden sind, werden sie in dem Muster verwendet. Das zweite Beispiel oben entspricht nicht `server1.example.com`, es entspricht aber `server07.example.com`. Um dies zu veranschaulichen, werden im folgenden Beispiel Bereiche verwendet, um die Definitionen der Gruppen `[usa]` und `[canada]` aus dem obigen Beispiel zu vereinfachen:

## Kapitel 2 | Implementieren eines Ansible-Playbooks

```
[usa]
washington[1:2].example.com

[canada]
ontario[01:02].example.com
```

## Verifizieren des Inventars

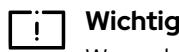
Im Zweifel führen Sie den Befehl `ansible` aus, um zu verifizieren, dass der Rechner im Inventar vorhanden ist:

```
[user@controlnode ~]$ ansible washington1.example.com --list-hosts
hosts (1):
  washington1.example.com
[user@controlnode ~]$ ansible washington01.example.com --list-hosts
[WARNING]: provided hosts list is empty, only localhost is available

hosts (0):
```

Sie können den folgenden Befehl ausführen, um alle Hosts in einer Gruppe aufzulisten:

```
[user@controlnode ~]$ ansible canada --list-hosts
hosts (2):
  ontario01.example.com
  ontario02.example.com
```



### Wichtig

Wenn das Inventar einen Host und eine Hostgruppe mit identischen Namen enthält, gibt der Befehl `ansible` eine Warnung aus und verwendet den Host als Ziel. Die Hostgruppe wird ignoriert.

Es gibt verschiedene Möglichkeiten, mit dieser Situation umzugehen: Am einfachsten ist es, sicherzustellen, dass Hostgruppen nicht dieselben Namen wie Hosts im Inventar verwenden.

## Überschreiben des Speicherorts für das Inventar

Die Datei `/etc/ansible/hosts` wird als statische Standardinventardatei des Systems betrachtet. Es ist jedoch gängige Praxis, diese Datei nicht zu verwenden, sondern in der Ansible-Konfigurationsdatei einen anderen Speicherort für Inventardateien zu definieren. Dies wird im nächsten Abschnitt behandelt.

Mit den Befehlen `ansible` und `ansible-playbook`, die Sie im weiteren Verlauf zur Ausführung von Ansible-Ad-hoc-Befehlen und -Playbooks verwenden, lässt sich der Speicherort einer Inventardatei auch in der Befehlszeile mit der Option `--inventory PATHNAME` oder `-i PATHNAME` angeben. Dabei ist `PATHNAME` der Pfad zur gewünschten Inventardatei.

## Definieren von Variablen im Inventar

In Hostinventardateien können Werte für Variablen angegeben werden, die in Playbooks verwendet werden. Diese Variablen gelten nur für bestimmte Hosts oder Hostgruppen.

Normalerweise ist es besser, diese *Inventarvariablen* in speziellen Verzeichnissen und nicht direkt in der Inventardatei zu definieren. Dieses Thema wird an anderer Stelle in diesem Kurs ausführlich behandelt.

## Beschreiben eines dynamischen Inventars

Ansible-Inventarinformationen lassen sich auch dynamisch mit Informationen generieren, die von externen Datenbanken bereitgestellt werden. Die Open Source Community hat eine Reihe von Skripten für das dynamische Inventar geschrieben, die über das Ansible-Upstream-Projekt erhältlich sind. Wenn diese Skripte Ihre Anforderungen nicht erfüllen, können Sie auch ein eigenes schreiben.

Ein Programm für dynamische Inventare könnte z. B. Ihren Red Hat Satellite Server oder Ihr Amazon EC2-Konto kontaktieren und dort gespeicherte Informationen zum Aufbau eines Ansible-Inventars verwenden. Weil das Programm so vorgeht, wenn Sie Ansible ausführen, kann es das Inventar mit aktuellen Informationen füllen, die vom Service bereitgestellt werden, während neue Hosts hinzugefügt und alte Hosts entfernt werden.



### Literaturhinweise

#### Inventory: Ansible-Dokumentation

[http://docs.ansible.com/ansible/2.9/user\\_guide/intro\\_inventory.html](http://docs.ansible.com/ansible/2.9/user_guide/intro_inventory.html)

## ► Angeleitete Übung

# Erstellen eines Ansible-Inventars

In dieser Übung erstellen Sie ein neues statisches Inventar, das Hosts und Gruppen enthält.

## Ergebnisse

Sie sollten in der Lage sein, standardmäßige und benutzerdefinierte statische Inventare zu erstellen.

## Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab deploy-inventory start` aus. Dieses Start-Skript stellt sicher, dass die verwalteten Hosts `servera`, `serverb`, `serverc` und `serverd` im Netzwerk erreichbar sind.

```
[student@workstation ~]$ lab deploy-inventory start
```

## Anweisungen

- ▶ 1. Ändern Sie `/etc/ansible/hosts` so, dass `servera.lab.example.com` als ein verwalteter Host enthalten ist.
  - 1.1. Fügen Sie `servera.lab.example.com` am Ende der standardmäßigen Inventardatei `/etc/ansible/hosts` hinzu.

```
[student@workstation ~]$ sudo vim /etc/ansible/hosts
...output omitted...
## db-[99:101]-node.example.com

servera.lab.example.com
```

- 1.2. Setzen Sie die Bearbeitung der Inventardatei `/etc/ansible/hosts` fort, indem Sie die Gruppe `[webservers]` am Ende der Datei mit dem Server `serverb.lab.example.com` als ein Gruppenmitglied hinzufügen. Speichern und beenden Sie, wenn Sie den Vorgang abgeschlossen haben.

```
...output omitted...
## db-[99:101]-node.example.com

servera.lab.example.com

[webservers]
serverb.lab.example.com
```

- ▶ 2. Verifizieren Sie die verwalteten Hosts in der Inventardatei `/etc/ansible/hosts`.

**Kapitel 2 |** Implementieren eines Ansible-Playbooks

- 2.1. Führen Sie den Befehl `ansible all --list-hosts` aus, um alle verwalteten Hosts in der Standardinventardatei aufzulisten.

```
[student@workstation ~]$ ansible all --list-hosts
hosts (2):
servera.lab.example.com
serverb.lab.example.com
```

- 2.2. Führen Sie den Befehl `ansible ungrouped --list-hosts` aus, damit nur die verwalteten Hosts aufgelistet werden, die nicht zu einer Gruppe gehören.

```
[student@workstation ~]$ ansible ungrouped --list-hosts
hosts (1):
servera.lab.example.com
```

- 2.3. Führen Sie den Befehl `ansible webservers --list-hosts` aus, damit nur die verwalteten Hosts aufgelistet werden, die zur Gruppe `webservers` gehören.

```
[student@workstation ~]$ ansible webservers --list-hosts
hosts (1):
serverb.lab.example.com
```

- 3. Erstellen Sie eine benutzerdefinierte statische Inventardatei mit dem Namen `inventory` im Arbeitsverzeichnis `/home/student/deploy-inventory`.

Informationen zu Ihren vier verwalteten Hosts werden in der folgenden Tabelle aufgelistet. Zu Verwaltungszwecken weisen Sie basierend auf dem Zweck des Hosts, der Stadt, in der er sich befindet, und der Bereitstellungsumgebung, zu der er gehört, die einzelnen Hosts mehreren Gruppen zu.

Darüber hinaus müssen Gruppen für die US-Städte (Raleigh und Mountain View) als untergeordnete Elemente der Gruppe `us` festgelegt werden, damit die Hosts in den Vereinigten Staaten als eine Gruppe verwaltet werden können.

### Serverinventarspezifikationen

Hostname	Zweck	Ort	Umgebung
servera.lab.example.com	Webserver	Raleigh	Entwicklung
serverb.lab.example.com	Webserver	Raleigh	Testing
serverc.lab.example.com	Webserver	Mountain View	Produktion
serverd.lab.example.com	Webserver	London	Produktion

- 3.1. Erstellen Sie das Arbeitsverzeichnis `/home/student/deploy-inventory`, und wechseln Sie in dieses Verzeichnis.

```
[student@workstation ~]$ mkdir ~/deploy-inventory
[student@workstation ~]$ cd ~/deploy-inventory
[student@workstation deploy-inventory]$
```

## Kapitel 2 | Implementieren eines Ansible-Playbooks

- 3.2. Erstellen Sie eine `inventory`-Datei im Arbeitsverzeichnis `/home/student/deploy-inventory`. Verwenden Sie die Tabelle „Serverinventarspezifikationen“ als Orientierungshilfe. Bearbeiten Sie die Datei `inventory`, und fügen Sie den folgenden Inhalt hinzu:

```
[webservers]
server[a:d].lab.example.com

[raleigh]
servera.lab.example.com
serverb.lab.example.com

[mountainview]
serverc.lab.example.com

[london]
serverd.lab.example.com

[development]
servera.lab.example.com

[testing]
serverb.lab.example.com

[production]
serverc.lab.example.com
serverd.lab.example.com

[us:children]
raleigh
mountainview
```

- 4. Verwenden Sie Variationen des Befehls `ansible host-or-group -i inventory --list-hosts`, um die verwalteten Hosts und Gruppen in der benutzerdefinierten Inventardatei `/home/student/deploy-inventory/inventory` zu verifizieren.



### Wichtig

In Ihrem Befehl `ansible` muss die Option `-i inventory` enthalten sein. Dadurch verwendet `ansible` die Datei `inventory` in Ihrem aktuellen Arbeitsverzeichnis anstatt der Systeminventardatei `/etc/ansible/hosts`.

- 4.1. Führen Sie den Befehl `ansible all -i inventory --list-hosts` aus, um alle verwalteten Hosts aufzulisten.

```
[student@workstation deploy-inventory]$ ansible all -i inventory --list-hosts
hosts (4):
servera.lab.example.com
serverb.lab.example.com
serverc.lab.example.com
serverd.lab.example.com
```

- 4.2. Führen Sie den Befehl `ansible ungrouped -i inventory --list-hosts` aus, um alle in der Inventardatei verwalteten Hosts, die kein Teil einer Gruppe sind, aufzulisten. In dieser Inventardatei gibt es keine ungruppierten verwalteten Hosts.

```
[student@workstation deploy-inventory]$ ansible ungrouped -i inventory \
> --list-hosts
[WARNING]: No hosts matched, nothing to do

hosts (0):
```

- 4.3. Führen Sie den Befehl `ansible development -i inventory --list-hosts` aus, um alle in der Gruppe `development` aufgeführten verwalteten Hosts auszulisten.

```
[student@workstation deploy-inventory]$ ansible development -i inventory \
> --list-hosts
hosts (1):
servera.lab.example.com
```

- 4.4. Führen Sie den Befehl `ansible testing -i inventory --list-hosts` aus, um alle in der Gruppe `testing` aufgeführten verwalteten Hosts auszulisten.

```
[student@workstation deploy-inventory]$ ansible testing -i inventory \
> --list-hosts
hosts (1):
serverb.lab.example.com
```

- 4.5. Führen Sie den Befehl `ansible production -i inventory --list-hosts` aus, um alle in der Gruppe `production` aufgeführten verwalteten Hosts aufzulisten.

```
[student@workstation deploy-inventory]$ ansible production -i inventory \
> --list-hosts
hosts (2):
serverc.lab.example.com
serverd.lab.example.com
```

- 4.6. Führen Sie den Befehl `ansible us -i inventory --list-hosts` aus, um alle in der Gruppe `us` aufgeführten verwalteten Hosts auszulisten.

```
[student@workstation deploy-inventory]$ ansible us -i inventory --list-hosts
hosts (3):
servera.lab.example.com
serverb.lab.example.com
serverc.lab.example.com
```

- 4.7. Sie sollten mit anderen Variationen experimentieren, um verwaltete Hosteinträge in der benutzerdefinierten Inventardatei zu bestätigen.

## Beenden

Führen Sie auf `workstation` das Skript `lab deploy-inventory finish` aus, um diese Übung zu bereinigen.

```
[student@workstation ~]$ lab deploy-inventory finish
```

Hiermit ist die angeleitete Übung beendet.

# Verwalten von Ansible-Konfigurationsdateien

---

## Ziele

Am Ende dieses Abschnitts sollten Sie in der Lage sein, zu beschreiben, wo Ansible-Konfigurationsdateien gespeichert sind, wie sie von Ansible ausgewählt werden und wie sie bearbeitet werden, um Änderungen an Standardeinstellungen vorzunehmen.

## Konfigurieren von Ansible

Das Verhalten von Ansible-Installationen kann durch Änderung der Einstellungen in der Ansible-Konfigurationsdatei angepasst werden. Ansible wählt die Konfigurationsdatei von einem der möglichen Speicherorte auf dem Kontrollknoten aus.

## Verwenden von /etc/ansible/ansible.cfg

Das `ansible`-Paket stellt unter `/etc/ansible/ansible.cfg` eine Basiskonfigurationsdatei bereit. Diese Datei wird verwendet, wenn keine andere Konfigurationsdatei gefunden wird.

## Verwenden von ~/.ansible.cfg

Ansible sucht im Benutzerverzeichnis nach der Datei `.ansible.cfg`. Diese Konfigurationsdatei wird anstelle der Datei `/etc/ansible/ansible.cfg` verwendet, wenn sie bereits vorhanden ist und im aktuellen Arbeitsverzeichnis keine `ansible.cfg`-Datei gefunden wird.

## Verwenden von ./ansible.cfg

Wenn die Datei `ansible.cfg` in dem Verzeichnis, in dem der Befehl `ansible` ausgeführt wird, bereits vorhanden ist, wird sie anstelle der globalen Datei bzw. der persönlichen Datei des Benutzers verwendet. Dies ermöglicht Administratoren, eine Verzeichnisstruktur zu erstellen, in der unterschiedliche Umgebungen oder Projekte in verschiedenen Verzeichnissen gespeichert werden, wobei jedes Verzeichnis eine Konfigurationsdatei mit individuellen Einstellungen enthält.



### Wichtig

Es wird empfohlen, die Datei `ansible.cfg` in einem Verzeichnis zu erstellen, in dem die Ansible-Befehle ausgeführt werden sollen. Dieses Verzeichnis sollte alle von Ihrem Ansible-Projekt genutzten Dateien enthalten, wie etwa ein Inventar und ein Playbook. Dies ist der am häufigsten verwendete Speicherort für Ansible-Konfigurationsdateien. Es ist ungewöhnlich, in der Praxis eine Datei `~/.ansible.cfg` oder `/etc/ansible/ansible.cfg` zu verwenden.

## Verwenden der Umgebungsvariablen ANSIBLE\_CONFIG

Sie können verschiedene Konfigurationsdateien verwenden, indem Sie diese in verschiedenen Verzeichnissen ablegen und anschließend im entsprechenden Verzeichnis Ansible-Befehle ausführen. Diese Methode kann jedoch zu Einschränkungen führen und erschwert durch eine zunehmende Anzahl der Konfigurationsdateien u. U. die Verwaltung. Eine flexiblere Option stellt die Definition des Speicherorts der Konfigurationsdatei mithilfe der Umgebungsvariablen

## Kapitel 2 | Implementieren eines Ansible-Playbooks

ANSIBLE\_CONFIG dar. Wenn diese Variable definiert wurde, verwendet Ansible die von der Variablen angegebene Konfigurationsdatei anstelle der zuvor genannten Konfigurationsdateien.

## Rangordnung von Konfigurationsdateien

Die Suchreihenfolge bei Konfigurationsdateien erfolgt in umgekehrter Reihenfolge zur vorausgegangenen Liste. Die erste Datei in der Suchreihenfolge wird von Ansible ausgewählt. Ansible verwendet nur die Konfigurationseinstellungen der ersten gefundenen Datei.

Jede von der Umgebungsvariablen ANSIBLE\_CONFIG angegebene Datei setzt alle anderen Konfigurationsdateien außer Kraft. Wenn diese Variable nicht festgelegt wurde, wird anschließend das Verzeichnis, in dem der Befehl ansible ausgeführt wurde, auf eine ansible.cfg-Datei überprüft. Wenn diese Datei nicht vorhanden ist, wird das Benutzerverzeichnis auf eine .ansible.cfg-Datei überprüft. Die globale Datei /etc/ansible/ansible.cfg wird nur verwendet, wenn keine andere Konfigurationsdatei gefunden wird. Falls die Konfigurationsdatei /etc/ansible/ansible.cfg nicht vorhanden ist, enthält Ansible Standardeinstellungen, die in diesem Fall verwendet werden.

Aufgrund der Vielzahl der Speicherorte, an denen Ansible-Konfigurationsdateien abgelegt werden können, ist es mitunter schwierig zu bestimmen, welche Konfigurationsdatei von Ansible verwendet wird. Sie können den Befehl ansible --version ausführen, um die installierte Version von Ansible und die verwendete Konfigurationsdatei eindeutig zu identifizieren.

```
[user@controlnode ~]$ ansible --version
ansible 2.9.21
  config file = /etc/ansible/ansible.cfg
...output omitted...
```

Eine andere Möglichkeit, die aktive Ansible-Konfigurationsdatei anzuzeigen, bietet die Option -v beim Ausführen von Ansible-Befehlen in der Befehlszeile.

```
[user@controlnode ~]$ ansible servers --list-hosts -v
Using /etc/ansible/ansible.cfg as config file
...output omitted...
```

Ansible verwendet nur Einstellungen aus der Konfigurationsdatei mit der höchsten Priorität. Auch wenn andere Dateien mit niedrigerer Rangordnung vorhanden sind, werden deren Einstellungen ignoriert und nicht mit den Einstellungen aus der ausgewählten Konfigurationsdatei kombiniert. Wenn Sie eine eigene Konfigurationsdatei in Anlehnung an die globale Konfigurationsdatei /etc/ansible/ansible.cfg erstellen möchten, müssen Sie daher alle gewünschten Einstellungen aus dieser Datei in die eigene Konfigurationsdatei auf Benutzerebene kopieren. Einstellungen, die in der Konfigurationsdatei auf Benutzerebene nicht definiert sind, bleiben auf die integrierten Standardeinstellungen festgelegt, selbst wenn sie in der globalen Konfigurationsdatei anderweitig festgelegt wurden.

## Verwalten von Einstellungen in der Konfigurationsdatei

Die Ansible-Konfigurationsdatei besteht aus mehreren Abschnitten, wobei jeder Abschnitt Einstellungen umfasst, die als Schlüssel-Wertpaare definiert sind. Die Abschnittstitel werden von eckigen Klammern umschlossen. Verwenden Sie für grundlegende Vorgänge die folgenden zwei Abschnitte:

- [defaults] Legt die Standardwerte für Ansible-Vorgänge fest

- [privilege\_escalation] Konfiguriert, wie Ansible auf verwalteten Hosts eine Rechteerweiterung durchführt

Beispielsweise sehen Sie im Folgenden eine typische `ansible.cfg`-Datei:

```
[defaults]
inventory = ./inventory
remote_user = user
ask_pass = false

[privilege_escalation]
become = true
become_method = sudo
become_user = root
become_ask_pass = false
```

Die Anweisungen in dieser Datei werden in der folgenden Tabelle erläutert:

### Ansible-Konfiguration

Direktive	Beschreibung
<code>inventory</code>	Gibt den Pfad zur Inventardatei an.
<code>remote_user</code>	Der Name des Benutzers, der sich auf den verwalteten Hosts anmelden soll. Wenn nichts angegeben ist, wird der Name des aktuellen Benutzers verwendet.
<code>ask_pass</code>	Gibt an, ob Sie zur Eingabe eines SSH-Passworts aufgefordert werden. Kann <code>false</code> sein, wenn die SSH-Authentifizierung mit einem öffentlichen Schlüssel zum Einsatz kommt.
<code>become</code>	Gibt an, ob der Benutzer nach dem Herstellen der Verbindung auf dem verwalteten Host (in der Regel zu <code>root</code> ) automatisch gewechselt werden soll. Dies kann auch durch ein Play festgelegt werden.
<code>become_method</code>	Art und Weise des Benutzerwechsels (in der Regel <code>sudo</code> , was dem Standard entspricht, möglich ist aber auch <code>su</code> ).
<code>become_user</code>	Der Benutzer, zu dem auf dem verwalteten Host gewechselt werden soll (in der Regel <code>root</code> , was dem Standard entspricht).
<code>become_ask_pass</code>	Gibt an, ob zur Eingabe eines Passworts für Ihre <code>become_method</code> aufgefordert werden soll. Der Standardwert lautet <code>false</code> .

## Konfigurieren von Verbindungen

Ansible muss erfahren, wie die Kommunikation mit den verwalteten Hosts erfolgen soll. Einer der häufigsten Gründe für eine Änderung der Konfigurationsdatei besteht darin, dass gesteuert werden soll, welche Methoden und Benutzer Ansible zur Administration verwalteter Hosts verwendet. Es werden u. a. folgende Informationen benötigt:

## Kapitel 2 | Implementieren eines Ansible-Playbooks

- Den Speicherort des Inventars, in dem die verwalteten Hosts und Hostgruppen aufgelistet werden
- Welches Verbindungsprotokoll zur Kommunikation mit den verwalteten Hosts verwendet werden soll (standardmäßig SSH) und ob zur Verbindung mit dem Server ein nicht standardmäßiger Netzwerkport benötigt wird
- Welcher Remote-Benutzer auf den verwalteten Hosts verwendet werden soll; dabei könnte es sich um `root` oder einen unprivilegierten Benutzer handeln
- Wenn der Remote-Benutzer unprivilegiert ist, benötigt Ansible die Information, ob versucht werden soll, Berechtigungen auf `root` auszuweiten, und wie dies geschehen soll (z. B. mithilfe von `sudo`)
- Soll für die Anmeldung oder den Erhalt von Berechtigungen zur Eingabe eines SSH-Passworts oder `sudo`-Passworts aufgefordert werden

## Speicherort von Inventaren

Die `inventory`-Anweisung im Abschnitt `[defaults]` kann unmittelbar auf eine statische Inventardatei oder auf ein Verzeichnis verweisen, das mehrere statische Inventardateien und Skripte für das dynamische Inventar enthält.

```
[defaults]
inventory = ./inventory
```

## Verbindungseinstellungen

Standardmäßig werden Ansible-Verbindungen mit den verwalteten Hosts über das SSH-Protokoll hergestellt. Die wichtigsten Parameter, die steuern, wie Ansible-Verbindungen mit den verwalteten Hosts hergestellt werden, sind im Abschnitt `[defaults]` festgelegt.

Standardmäßig versucht Ansible, mithilfe des Benutzernamens des lokalen Benutzers, der die Ansible-Befehle ausführt, eine Verbindung mit dem verwalteten Host herzustellen. Um einen anderen Remote-Benutzer anzugeben, legen Sie den Parameter `remote_user` auf den jeweiligen Benutzernamen fest.

Wenn für den lokalen Benutzer, der Ansible ausführt, private SSH-Schlüssel konfiguriert wurden, mit denen er sich auf den verwalteten Hosts als Remote-Benutzer authentifizieren kann, meldet sich Ansible automatisch an. Andernfalls kann Ansible durch Festlegen der Anweisung `ask_pass = true` so konfiguriert werden, dass der lokale Benutzer zur Eingabe des Passworts aufgefordert wird, das der Remote-Benutzer verwendet.

```
[defaults]
inventory = ./inventory

remote_user = root
ask_pass = true
```

Gesetzt den Fall, dass Sie einen Linux-Kontrollknoten und OpenSSH auf Ihren verwalteten Hosts verwenden und sich mit einem Passwort als Remote-Benutzer anmelden können, dann können Sie wahrscheinlich die schlüsselbasierte SSH-Authentifizierung einrichten, mit der Sie `ask_pass = false` festlegen könnten.

## Kapitel 2 | Implementieren eines Ansible-Playbooks

In einem ersten Schritt muss sichergestellt werden, dass in `~/.ssh` ein SSH-Schlüsselpaar für den Benutzer auf dem Kontrollknoten konfiguriert ist. Hierzu kann der Befehl `ssh-keygen` ausgeführt werden.

Wenn nur ein einziger verwalteter Host vorhanden ist, können Sie wie folgt Ihren öffentlichen Schlüssel auf dem verwalteten Host installieren und den Befehl `ssh-copy-id` ausführen, um Ihre lokale Datei `~/.ssh/known_hosts` mit dem zugehörigen Hostschlüssel zu füllen:

```
[user@controlnode ~]$ ssh-copy-id root@web1.example.com
The authenticity of host 'web1.example.com (192.168.122.181)' can't be
established.
ECDSA key fingerprint is 70:9c:03:cd:de:ba:2f:11:98:fa:a0:b3:7c:40:86:4b.
Are you sure you want to continue connecting (yes/no)? yes
/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
root@web1.example.com's password:

Number of key(s) added: 1

Now try logging into the machine, with: "ssh 'root@web1.example.com'"
and check to make sure that only the key(s) you wanted were added.
```



### Anmerkung

Sie können auch ein Ansible-Playbook verwenden, um Ihren öffentlichen Schlüssel mithilfe des Moduls `authorized_key` im Konto `remote_user` auf *allen* verwalteten Hosts bereitzustellen.

In diesem Kurs wurden Ansible-Playbooks noch nicht ausführlich behandelt. Ein Play, mit dem sichergestellt wird, dass Ihr öffentlicher Schlüssel in den `root`-Konten auf den verwalteten Hosts bereitgestellt wird, könnte folgende Zeilen enthalten:

```
- name: Public key is deployed to managed hosts for Ansible
hosts: all

tasks:
  - name: Ensure key is in root's ~/.ssh/authorized_hosts
    authorized_key:
      user: root
      state: present
      key: '{{ item }}'
    with_file:
      - ~/.ssh/id_rsa.pub
```

Da für den verwalteten Host noch keine schlüsselbasierte SSH-Authentifizierung konfiguriert wurde, müssten Sie das Playbook ausführen und hierzu den Befehl `ansible-playbook` mit der Option `--ask-pass` verwenden, damit sich der Befehl als Remote-Benutzer authentifizieren kann.

## Ausweiten von Berechtigungen

Aus Gründen der Sicherheit und der Überwachung muss Ansible Verbindungen mit Remote-Hosts möglicherweise als nicht berechtigter Benutzer herstellen, bevor die Rechte erweitert werden, um administrativen Zugriff als `root` zu erhalten. Dies kann im Abschnitt `[privilege_escalation]` der Ansible-Konfigurationsdatei eingerichtet werden.

Um die Rechteerweiterung standardmäßig zu aktivieren, legen Sie in der Konfigurationsdatei die Anweisung `become = true` fest. Selbst wenn diese standardmäßig festgelegt wird, gibt es verschiedene Möglichkeiten, sie bei Ausführung von Ad-hoc-Befehlen oder Ansible-Playbooks außer Kraft zu setzen. (Es sind z. B. Situationen denkbar, in denen Sie Aufgaben oder Plays ausführen möchten, bei denen die Rechte nicht erweitert werden.)

Mit der Anweisung `become_method` wird angegeben, wie Rechte erweitert werden sollen. Es sind mehrere Optionen verfügbar, aber standardmäßig wird `sudo` verwendet. In gleicher Weise wird mit der Anweisung `become_user` der Benutzer angegeben, auf den erweitert werden soll, aber standardmäßig wird `root` verwendet.

Wenn der gewählte `become_method`-Mechanismus verlangt, dass der Benutzer ein Passwort eingibt, um die Rechte zu erweitern, kann in der Konfigurationsdatei die Anweisung `become_ask_pass = true` festgelegt werden.



### Anmerkung

Unter Red Hat Enterprise Linux 7 können aufgrund der Standardkonfiguration von `/etc/sudoers` alle Benutzer in der Gruppe `wheel` `sudo` verwenden, um nach der Eingabe ihres Passworts `root` zu werden.

Eine Methode, einem Benutzer („`someuser`“ in dem folgenden Beispiel) zu ermöglichen, mithilfe von `sudo` ohne ein Passwort `root` zu werden, besteht darin, eine Datei mit den entsprechenden Anweisungen im Verzeichnis `/etc/sudoers.d` (im Besitz von `root`, mit der oktalen Berechtigung 0400) zu installieren:

```
## password-less sudo for Ansible user
someuser ALL=(ALL) NOPASSWD:ALL
```

Überdenken Sie die Auswirkungen, die Ihre jeweils gewählte Methode zur Rechteerweiterung für die Sicherheit hat. Verschiedene Organisationen und Bereitstellungen müssen möglicherweise unterschiedliche Kompromisse eingehen.

Die folgende `ansible.cfg`-Beispieldatei setzt voraus, dass Sie als `someuser` mit schlüsselbasierter SSH-Authentifizierung Verbindungen mit den verwalteten Hosts herstellen können und dass `someuser` mithilfe von `sudo` Befehle als `root` ausführen kann, ohne ein Passwort einzugeben:

```
[defaults]
inventory = ./inventory
remote_user = someuser
ask_pass = false

[privilege_escalation]
become = true
```

```
become_method = sudo
become_user = root
become_ask_pass = false
```

## Nicht-SSH-Verbindungen

Das Protokoll, mit dem Ansible eine Verbindung mit verwalteten Hosts herstellt, ist standardmäßig auf `smart` festgelegt, wodurch die effizienteste Methode zur Verwendung von SSH festgelegt wird. Dies kann in unterschiedlicher Weise auf andere Werte festgelegt werden.

Es gibt z. B. eine Ausnahme zu der Regel, dass standardmäßig SSH verwendet wird. Wenn in Ihrem Inventar `localhost` nicht vorhanden ist, richtet Ansible den Eintrag `implicit localhost` ein, damit Sie Ad-hoc-Befehle und Playbooks ausführen können, die auf `localhost` abzielen. Dieser spezielle Inventareintrag ist in den Hostgruppen `all` und `ungrouped` nicht enthalten. Zusätzlich verwendet Ansible zur Verbindung mit diesem Host standardmäßig den speziellen Verbindungstyp `local` an Stelle des `smart` SSH-Verbindungstyps.

```
[user@controlnode ~]$ ansible localhost --list-hosts
[WARNING]: provided hosts list is empty, only localhost is available

hosts (1):
  localhost
```

Mit dem Verbindungstyp `local` wird die Einstellung `remote_user` ignoriert und Befehle werden direkt auf dem lokalen System aufgeführt. Bei Verwendung der Rechteerweiterung wird `sudo` mit dem Benutzerkonto ausgeführt, das den Ansible-Befehl ausgeführt hat, statt mit `remote_user`. Dies kann zu Missverständnissen führen, wenn die beiden Benutzer über unterschiedliche `sudo`-Berechtigungen verfügen.

Wenn Sie sicherstellen möchten, dass die Verbindung mit `localhost` wie bei anderen verwalteten Hosts mithilfe von SSH hergestellt wird, besteht eine Methode darin, diesen Host in Ihrem Inventar aufzuführen. Auf diese Weise wird er jedoch in die Gruppen `all` und `ungrouped` aufgenommen, was möglicherweise nicht gewünscht ist.

Eine weitere Methode besteht darin, das Protokoll zu ändern, das zur Verbindung mit `localhost` verwendet wird. Es empfiehlt sich, die Hostvariable `ansible_connection` für `localhost` festzulegen. Erstellen Sie hierzu in dem Verzeichnis, in dem Sie Ansible-Befehle ausführen, das Unterverzeichnis `host_vars`. Erstellen Sie in diesem Unterverzeichnis eine Datei mit dem Namen `localhost`, welche die Zeile `ansible_connection: smart` enthält. So wird sichergestellt, dass für `localhost` das Verbindungsprotokoll `smart` (SSH) anstelle von `local` verwendet wird.

Dies kann auch in der entgegengesetzten Richtung genutzt werden. Wenn `127.0.0.1` in Ihrem Inventar aufgeführt wird, stellen Sie die Verbindung mit diesem Host standardmäßig mithilfe von `smart` her. Sie können auch eine Datei vom Typ `host_vars/127.0.0.1` erstellen, welche die Zeile `ansible_connection: local` enthält und stattdessen `local` verwendet.

Hostvariablen werden im weiteren Verlauf dieses Kurses ausführlicher behandelt.



### Anmerkung

Sie können auch *Gruppenvariablen* verwenden, um den Verbindungstyp für eine gesamte Hostgruppe zu ändern. Dies lässt sich bewerkstelligen, indem Dateien mit demselben Namen wie die Gruppe im Verzeichnis `group_vars` platziert werden und sichergestellt wird, dass diese Dateien Einstellungen für die Verbindungsvariablen enthalten.

Vielleicht sollen z. B. Ihre gesamten verwalteten Microsoft Windows-Hosts für Verbindungen das Protokoll `winrm` und den Port 5986 verwenden. Um dies zu konfigurieren, könnten Sie diese verwalteten Hosts in der Gruppe `windows` platzieren und anschließend eine Datei mit dem Namen `group_vars/windows` erstellen, die die folgenden Zeilen enthält:

```
ansible_connection: winrm
ansible_port: 5986
```

## Kommentare in Konfigurationsdateien

Es gibt zwei Kommentarzeichen, die in Ansible-Konfigurationsdateien zulässig sind: das Rauten- oder Nummernzeichen (#) und das Semikolon (;).

Mit dem Nummernzeichen am Anfang einer Zeile wird die gesamte Zeile auskommentiert. Es darf nicht in einer Zeile mit einer Anweisung verwendet werden.

Mit dem Semikolon wird der gesamte Zeileninhalt rechts neben dem Zeichen auskommentiert. Es darf in einer Zeile mit einer Anweisung verwendet werden, solange diese Anweisung links vom Zeichen platziert ist.



### Literaturhinweise

Manpages `ansible(1)`, `ansible-config(1)`, `ssh-keygen(1)` und `ssh-copy-id(1)`

#### Configuration file: Ansible-Dokumentation

[https://docs.ansible.com/ansible/2.9/installation\\_guide/intro\\_configuration.html](https://docs.ansible.com/ansible/2.9/installation_guide/intro_configuration.html)

## ► Angeleitete Übung

# Verwalten von Ansible-Konfigurationsdateien

In dieser Übung passen Sie Ihre Ansible-Umgebung an, indem Sie eine Ansible-Konfigurationsdatei bearbeiten.

### Ergebnisse

Sie sollten in der Lage sein, eine Konfigurationsdatei zu erstellen, um Ihre Ansible-Umgebung mit persistenten, benutzerdefinierten Einstellungen zu konfigurieren.

### Bevor Sie Beginnen

Melden Sie sich als student mit dem Passwort student bei workstation an.

Führen Sie auf workstation den Befehl `lab deploy-manage start` aus. Dieses Skript stellt sicher, dass der verwaltete Host servera im Netzwerk erreichbar ist.

```
[student@workstation ~]$ lab deploy-manage start
```

### Anweisungen

- 1. Erstellen Sie das Verzeichnis `/home/student/deploy-manage`, in dem die Dateien für diese Übung gespeichert werden. Wechseln Sie in dieses neu erstellte Verzeichnis.

```
[student@workstation ~]$ mkdir ~/deploy-manage  
[student@workstation ~]$ cd ~/deploy-manage  
[student@workstation deploy-manage]$
```

- 2. Verwenden Sie im Verzeichnis `/home/student/deploy-manage` einen Texteditor, um mit der Bearbeitung der neuen Datei `ansible.cfg` zu beginnen.

Erstellen Sie in dieser Datei den Abschnitt `[defaults]`. Fügen Sie in diesem Abschnitt eine Zeile hinzu, in der die Datei `./inventory` mithilfe der Anweisung `inventory` als Standardinventar festgelegt wird.

```
[defaults]  
inventory = ./inventory
```

Speichern Sie Ihre Arbeit und beenden Sie den Texteditor.

- 3. Verwenden Sie im Verzeichnis `/home/student/deploy-manage` einen Texteditor, um mit der Bearbeitung der neuen statischen Inventardatei `inventory` zu beginnen.

Das statische Inventar sollte vier Hostgruppen enthalten:

- `[myself]` sollte den Host `localhost` enthalten.
- `[intranetweb]` sollte den Host `servera.lab.example.com` enthalten.
- `[internetweb]` sollte den Host `serverb.lab.example.com` enthalten.

## Kapitel 2 | Implementieren eines Ansible-Playbooks

- [web] sollte die Hostgruppen `intranetweb` und `internetweb` enthalten.

- 3.1. Erstellen Sie in `/home/student/deploy-manage/inventory` die Hostgruppe `myself`, indem Sie die folgenden Zeilen hinzufügen:

```
[myself]
localhost
```

- 3.2. Erstellen Sie in `/home/student/deploy-manage/inventory` die Hostgruppe `intranetweb`, indem Sie die folgenden Zeilen hinzufügen:

```
[intranetweb]
servera.lab.example.com
```

- 3.3. Erstellen Sie in `/home/student/deploy-manage/inventory` die Hostgruppe `internetweb`, indem Sie die folgenden Zeilen hinzufügen:

```
[internetweb]
serverb.lab.example.com
```

- 3.4. Erstellen Sie in `/home/student/deploy-manage/inventory` die Hostgruppe `web`, indem Sie die folgenden Zeilen hinzufügen:

```
[web:children]
intranetweb
internetweb
```

- 3.5. Bestätigen Sie, dass Ihre Datei `inventory` schließlich folgendermaßen aussieht:

```
[myself]
localhost

[intranetweb]
servera.lab.example.com

[internetweb]
serverb.lab.example.com

[web:children]
intranetweb
internetweb
```

Speichern Sie Ihre Arbeit und beenden Sie den Texteditor.

- 4. Führen Sie den Befehl `ansible` mit der Option `--list-hosts` aus, um die Konfiguration der Hostgruppen in Ihrer Inventardatei zu testen. Damit wird keine tatsächliche Verbindung mit diesen Hosts hergestellt.

```
[student@workstation deploy-manage]$ ansible myself --list-hosts
  hosts (1):
    localhost
[student@workstation deploy-manage]$ ansible intranetweb --list-hosts
```

```
hosts (1):
    servera.lab.example.com
[student@workstation deploy-manage]$ ansible internetweb --list-hosts
hosts (1):
    serverb.lab.example.com
[student@workstation deploy-manage]$ ansible web --list-hosts
hosts (2):
    servera.lab.example.com
    serverb.lab.example.com
[student@workstation deploy-manage]$ ansible all --list-hosts
hosts (3):
    localhost
    servera.lab.example.com
    serverb.lab.example.com
```

- 5. Öffnen Sie die Datei /home/student/deploy-manage/ansible.cfg in einem Texteditor. Fügen Sie den Abschnitt [privilege\_escalation] hinzu, um Ansible so zu konfigurieren, dass automatisch der Befehl sudo ausgeführt wird, um von student zu root zu wechseln, wenn Aufgaben auf den verwalteten Hosts ausgeführt werden. Ansible sollte außerdem so konfiguriert werden, dass Sie zur Eingabe des Passworts aufgefordert werden, das von student für den Befehl sudo verwendet wird.
- 5.1. Erstellen Sie in der Konfigurationsdatei /home/student/deploy-manage/ansible.cfg den Abschnitt [privilege\_escalation], indem Sie den folgenden Eintrag hinzufügen:

```
[privilege_escalation]
```

- 5.2. Aktivieren Sie die Rechteerweiterung, indem Sie die Anweisung become auf true festlegen.

```
become = true
```

- 5.3. Legen Sie die Rechteerweiterung für die Ausführung des Befehls sudo fest, indem Sie die Anweisung become\_method auf sudo festlegen.

```
become_method = sudo
```

- 5.4. Legen Sie den Benutzer für die Rechteerweiterung fest, indem Sie die Anweisung become\_user auf root festlegen.

```
become_user = root
```

- 5.5. Aktivieren Sie die Aufforderung zur Eingabe des Passworts für die Rechteerweiterung, indem Sie die Anweisung become\_ask\_pass auf true festlegen.

```
become_ask_pass = true
```

- 5.6. Bestätigen Sie, dass die vollständige Datei ansible.cfg folgendermaßen aussieht:

```
[defaults]
inventory = ./inventory

[privilegeEscalation]
become = true
becomeMethod = sudo
becomeUser = root
becomeAskPass = true
```

Speichern Sie Ihre Arbeit und beenden Sie den Texteditor.

- 6. Führen Sie den Befehl `ansible --list-hosts` erneut aus, um zu verifizieren, dass Sie jetzt aufgefordert werden, das `sudo`-Passwort einzugeben.

Wenn Sie zur Eingabe des `sudo`-Passworts aufgefordert werden, geben Sie `student` ein, auch wenn es für diesen Probelauf nicht verwendet wird.

```
[student@workstation deploy-manage]$ ansible intranetweb --list-hosts
BECOME password: student
hosts (1):
servera.lab.example.com
```

## Beenden

Führen Sie auf `workstation` das Skript `lab deploy-manage finish` aus, um diese Übung zu bereinigen.

```
[student@workstation ~]$ lab deploy-manage finish
```

Hiermit ist die angeleitete Übung beendet.

# Ausführen von Ad-hoc-Befehlen

## Ziele

Nach Abschluss dieses Abschnitts sollten Sie in der Lage sein, eine einzelne Ansible-Automatisierungsaufgabe mithilfe eines Ad-hoc-Befehls auszuführen sowie einige Anwendungsfälle für Ad-hoc-Befehle zu beschreiben.

## Ausführen von Ad-hoc-Befehlen mit Ansible

Ein *Ad-hoc-Befehl* stellt eine Möglichkeit dar, schnell eine einzelne Ansible-Aufgabe auszuführen, die Sie nicht speichern müssen, um sie später erneut auszuführen. Es handelt sich um einfache Online-Operationen, die ohne Schreiben eines Playbooks ausgeführt werden können.

Ad-hoc-Befehle eignen sich für schnelle Tests und Änderungen. Sie können z. B. einen Ad-hoc-Befehl verwenden, um sicherzustellen, dass in der Datei `/etc/hosts` auf einer Gruppe von Servern eine bestimmte Zeile vorhanden ist. Mit einem anderen Ad-hoc-Befehl könnten Sie einen Service effizient auf vielen verschiedenen Rechnern neu starten oder sicherstellen, dass ein bestimmtes Softwarepaket auf dem neuesten Stand ist.

Ad-hoc-Befehle sind sehr nützlich zur schnellen Ausführung einfacher Aufgaben mit Ansible. Sie haben ihre Grenzen und im Allgemeinen sollten Sie Ansible-Playbooks verwenden, um das gesamte Potenzial von Ansible ausschöpfen zu können. In vielen Situationen sind Ad-hoc-Befehle jedoch genau das Werkzeug, das Sie brauchen, um schnell einfache Aufgaben durchzuführen.

## Ausführen von Ad-hoc-Befehlen

Verwenden Sie den Befehl `ansible`, um Ad-hoc-Befehle auszuführen:

```
ansible host-pattern -m module [-a 'module arguments'] [-i inventory]
```

Das Argument *host-pattern* wird verwendet, um die verwalteten Hosts anzugeben, auf denen der Ad-hoc-Befehl ausgeführt werden soll. Dabei könnte es sich um einen bestimmten verwalteten Host oder eine Hostgruppe im Inventar handeln. Sie haben bereits gesehen, wie dieses Argument in Verbindung mit der Option `--list-hosts` verwendet wird, um die Hosts anzuzeigen, die mit einem bestimmten Host-Pattern übereinstimmen. Auch haben Sie bereits festgestellt, dass Sie mithilfe der Option `-i` einen anderen Inventarspeicherort als den Standardort aus der aktuellen Ansible-Konfigurationsdatei angeben können, damit er verwendet wird.

Die Option `-m` nimmt als Argument den Namen des *Moduls* an, das Ansible auf den Zielhosts ausführen soll. Module sind kleine Programme, die ausgeführt werden, um Ihre Aufgabe zu implementieren. Manche Module benötigen keine zusätzlichen Informationen, aber für andere sind zusätzliche Argumente erforderlich, um die Details ihrer Funktion anzugeben. Die Option `-a` verwendet eine Liste dieser Argumente als Zeichenfolge in Anführungszeichen.

Einer der einfachsten Ad-hoc-Befehle verwendet das Modul `ping`. Dieses Modul führt keinen ICMP-Ping durch, sondern überprüft, ob Python-basierte Module auf verwalteten Hosts ausgeführt werden können. Mit dem folgenden Ad-hoc-Befehl wird z. B. ermittelt, ob alle verwalteten Hosts im Inventar Standardmodule ausführen können:

```
[user@controlnode ~]$ ansible all -m ping
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
```

## Durchführen von Aufgaben mit Modulen in Ad-hoc-Befehlen

Module sind die Werkzeuge, mit denen Ad-hoc-Befehle Aufgaben durchführen. Ansible stellt mehrere Hundert Module bereit, die verschiedene Funktionen erfüllen. Normalerweise lässt sich als Teil der Standardinstallation ein getestetes spezielles Modul finden, das die benötigte Funktion erfüllt.

Mit dem Befehl `ansible-doc -l` werden alle Module aufgelistet, die auf einem System installiert sind. Sie können `ansible-doc` verwenden, um die Dokumentation zu bestimmten Modulen nach Namen anzuzeigen und Informationen darüber zu suchen, welche Argumente von den Modulen als Optionen verwendet werden. Mit dem folgenden Befehl wird z. B. Dokumentation für das Modul `ping` angezeigt:

```
[user@controlnode ~]$ ansible-doc ping
> PING      (/usr/lib/python3.6/site-packages/ansible/modules/system/ping.py)

A trivial test module, this module always returns 'pong' on successful
contact. It does not make sense in playbooks, but it is useful from `/usr/bin/
ansible` to
    verify the ability to login and that a usable Python is configured. This
is NOT ICMP ping, this is just a trivial test module that requires Python on the
remote-node. For Windows targets, use the [win_ping] module instead. For
Network targets, use the [net_ping] module instead.

* This module is maintained by The Ansible Core Team
OPTIONS (= is mandatory):

- data
    Data to return for the `ping` return value.
    If this parameter is set to `crash`, the module will cause an exception.
    [Default: pong]
    type: str

SEE ALSO:
* Module net_ping
    The official documentation on the net_ping module.
    https://docs.ansible.com/ansible/2.9/modules/net\_ping\_module.html
* Module win_ping
    The official documentation on the win_ping module.
    https://docs.ansible.com/ansible/2.9/modules/win\_ping\_module.html
```

AUTHOR: Ansible Core Team, Michael DeHaan

METADATA:

```
status:
  - stableinterface
  supported_by: core
```

## EXAMPLES:

```
# Test we can logon to 'webservers' and execute python with json lib.
# ansible webservers -m ping

# Example from an Ansible Playbook
- ping:

# Induce an exception to see what happens
- ping:
    data: crash
```

## RETURN VALUES:

```
ping:
  description: value provided with the data parameter
  returned: success
  type: str
  sample: pong
```

Weitere Informationen zu Modulen finden Sie in der Online-Dokumentation zu Ansible unter [http://docs.ansible.com/ansible/2.9/modules/modules\\_by\\_category.html](http://docs.ansible.com/ansible/2.9/modules/modules_by_category.html).

In der folgenden Tabelle sind einige nützliche Module als Beispiele aufgeführt. Es gibt noch viele andere.

**Ansible-Module**

Modulkategorie	Module
Dateimodule	<ul style="list-style-type: none"> <li>copy: Kopiert eine lokale Datei auf den verwalteten Host</li> <li>file: Legt Berechtigungen und andere Dateieigenschaften fest</li> <li>lineinfile: Stellt sicher, dass sich eine bestimmte Zeile (nicht) in einer Datei befindet</li> <li>synchronize: Synchronisiert Inhalt mit rsync</li> </ul>
Softwarepaketmodule	<ul style="list-style-type: none"> <li>package: Verwaltet Pakete mit dem automatisch erkannten Paket-Manager, der für das Betriebssystem nativ ist</li> <li>yum: Verwalten Sie Pakete mit dem YUM-Paket-Manager</li> <li>apt: Verwaltet Pakete mit dem APT-Paket-Manager</li> <li>dnf: Verwaltet Pakete mit dem DNF-Paket-Manager</li> <li>gem: Verwaltet Ruby-Gems</li> <li>pip: Verwalten Sie Python-Pakete von PyPI.</li> </ul>

Modulkategorie	Module
Systemmodule	<ul style="list-style-type: none"> <li><b>firewalld</b>: Verwaltet arbiträre Ports und Services mit <code>firewalld</code></li> <li><b>reboot</b>: Starten Sie einen Rechner neu.</li> <li><b>service</b>: Verwaltet Services</li> <li><b>user</b>: Fügt Benutzerkonten hinzu, entfernt und verwaltet diese</li> </ul>
Net Tools-Module	<ul style="list-style-type: none"> <li><b>get_url</b>: Laden Sie Dateien über HTTP, HTTPS oder FTP herunter.</li> <li><b>nmcli</b>: Verwalten Sie Netzwerke.</li> <li><b>uri</b>: Interagiert mit Webservices</li> </ul>

Die meisten Module verwenden Argumente. Die Liste mit den Argumenten, die für die verschiedenen Module zur Verfügung stehen, finden Sie in der Dokumentation zu dem jeweiligen Modul. Ad-hoc-Befehle übergeben Argumente mithilfe der Option `-a` an Module. Wenn kein Argument benötigt wird, lassen Sie die Option `-a` bei Ad-hoc-Befehlen aus. Wenn mehrere Argumente angegeben werden müssen, stellen Sie sie – in Anführungszeichen gesetzt und durch Leerzeichen getrennt – in einer Liste bereit.

Der folgende Ad-hoc-Befehl z. B. verwendet das Modul `user`, um sicherzustellen, dass der Benutzer `newbie` vorhanden ist und auf `servera.lab.example.com` die UID 4000 besitzt:

```
[user@controlnode ~]$ ansible -m user -a 'name=newbie uid=4000 state=present' \
> servera.lab.example.com
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": true,
    "comment": "",
    "createhome": true,
    "group": 4000,
    "home": "/home/newbie",
    "name": "newbie",
    "shell": "/bin/bash",
    "state": "present",
    "system": false,
    "uid": 4000
}
```

Die meisten Module sind *idempotent*. Dies bedeutet, dass sie mehrere Male sicher ausgeführt werden können, und nichts unternehmen, wenn das System sich bereits im richtigen Status befindet. Wenn Sie beispielsweise den vorherigen Ad-hoc-Befehl erneut ausführen, sollte keine Änderung gemeldet werden:

```
[user@controlnode ~]$ ansible -m user -a 'name=newbie uid=4000 state=present' \
> servera.lab.example.com
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "append": false,
    "changed": false
```

```
"comment": "",  
"group": 4000,  
"home": "/home/newbie",  
"move_home": false,  
"name": "newbie",  
"shell": "/bin/bash",  
"state": "present",  
"uid": 4000  
}
```

## Ausführen von arbiträren Befehlen auf verwalteten Hosts

Das Modul `command` ermöglicht es Administratoren, in der Befehlszeile verwalteter Hosts beliebige Befehle auszuführen. Der auszuführende Befehl wird mithilfe der Option `-a` als Argument für das Modul angegeben. Beispiel: Mit dem folgenden Befehl wird der Befehl `hostname` auf den verwalteten Hosts ausgeführt, auf die über das Host-Pattern `mymanagedhosts` verwiesen wird.

```
[user@controlnode ~]$ ansible mymanagedhosts -m command -a /usr/bin/hostname  
host1.lab.example.com | CHANGED | rc=0 >>  
host1.lab.example.com  
host2.lab.example.com | CHANGED | rc=0 >>  
host2.lab.example.com
```

Im vorherigen Beispiel gab der Ad-hoc-Befehl für jeden verwalteten Host zwei Ausgabezeilen zurück. In der ersten Zeile wird ein Statusbericht mit dem Namen des verwalteten Hosts angezeigt, auf den der Ad-hoc-Befehl angewendet wurde, sowie das Ergebnis des Befehls. Die zweite Zeile enthält die Ausgabe des Befehls, der remote mithilfe des Ansible-Moduls `command` ausgeführt wurde.

Für eine bessere Lesbarkeit und Analyse der Ausgabe von Ad-hoc-Befehlen haben Administratoren die Möglichkeit, zu jedem Befehl, der auf einem verwalteten Host ausgeführt wird, nur eine einzige Ausgabezeile anzuzeigen. Mit der Option `-o` können Sie die Ausgabe der Ad-hoc-Befehle von Ansible im einzeiligen Format anzeigen.

```
[user@controlnode ~]$ ansible mymanagedhosts -m command -a /usr/bin/hostname -o  
host1.lab.example.com | CHANGED | rc=0 >> (stdout) host1.lab.example.com  
host2.lab.example.com | CHANGED | rc=0 >> (stdout) host2.lab.example.com
```

Das Modul `command` ermöglicht es Administratoren, Remote-Befehle auf verwalteten Hosts schnell auszuführen. Diese Befehle werden von der Shell auf den verwalteten Hosts nicht verarbeitet. Daher können Sie mit diesen Befehlen nicht auf Shell-Umgebungsvariablen zugreifen oder Shell-Vorgänge wie Umleitungen und Piping durchführen.



### Anmerkung

Wenn in einem Ad-hoc-Befehl das mit der Option `-m` zu verwendende Modul nicht angegeben ist, verwendet Ansible standardmäßig das Modul `command`.

In Situationen, in denen eine Shell-Verarbeitung von Befehlen erforderlich ist, können Administratoren das Modul `shell` verwenden. Wie beim Modul `command` übergeben Sie die auszuführenden Befehle als Argumente an das Modul in einem Ad-hoc-Befehl. Ansible führt den Befehl daraufhin remote auf den verwalteten Hosts aus. Im Gegensatz zum Modul `command`

## Kapitel 2 | Implementieren eines Ansible-Playbooks

werden die Befehle über eine Shell auf den verwalteten Hosts verarbeitet. Daher ist der Zugriff auf Shell-Umgebungsvariablen möglich und es können Shell-Vorgänge wie Umleitungen und Piping genutzt werden.

Das folgende Beispiel veranschaulicht den Unterschied zwischen den Modulen `command` und `shell`. Wenn Sie versuchen, den integrierten Bash-Befehl `set` mit diesen beiden Modulen auszuführen, ist er nur mit dem `shell`-Modul erfolgreich.

```
[user@controlnode ~]$ ansible localhost -m command -a set
localhost | FAILED | rc=2 >>
[Errno 2] No such file or directory
[user@controlnode ~]$ ansible localhost -m shell -a set
localhost | CHANGED | rc=0 >>
BASH=/bin/sh
BASHOPTS=cmdhist:extquote:force_fignore:hostcomplete:interactive_comments:progcomp:promptvars:sourcepath
BASH_ALIASES=()
...output omitted...
```

Für die Module `command` und `shell` wird eine funktionierende Python-Installation auf dem verwalteten Host benötigt. Mit dem dritten Modul `raw` können Befehle unter Umgehung des Modulsubsystems direkt in der Remote-Shell ausgeführt werden. Dies ist bei der Verwaltung von Systemen nützlich, auf denen Python nicht installiert werden kann (z. B. ein Netzwerk-Router). Es kann auch verwendet werden, um Python auf einem Host zu installieren.



### Wichtig

In den meisten Fällen ist es die empfohlene Vorgehensweise, die Module `command`, `shell` und `raw` zur Ausführung von Befehlen nicht zu verwenden.

Die meisten anderen Module sind idempotent und können Änderungen automatisch nachverfolgen. Sie können den Status von Systemen testen und unternehmen nichts, wenn diese Systeme sich bereits im richtigen Status befinden. Im Gegensatz dazu ist es wesentlich komplizierter, die Module zur Ausführung von Befehlen auf idempotente Weise zu verwenden. Auf solche Module angewiesen zu sein, macht es für Sie schwieriger, darauf zu vertrauen, dass die erneute Ausführung eines Ad-hoc-Befehls oder Playbooks nicht zu einem unerwarteten Fehler führt. Wenn das Modul `shell` oder `command` ausgeführt wird, wird je nach dem angenommenen Rechnerstatus für gewöhnlich der Status `CHANGED` gemeldet.

Manchmal sind Module zur Ausführung von Befehlen nützliche Werkzeuge und eine gute Lösung für ein Problem. Wenn Sie sie verwenden müssen, ist es wahrscheinlich am besten, zuerst das Modul `command` zu verwenden und erst dann auf das Modul `shell` oder `raw` zurückzugreifen, wenn Sie deren spezielle Funktionen benötigen.

## Konfigurieren von Verbindungen für Ad-hoc-Befehle

Die Anweisungen für die Verbindung mit verwalteten Hosts und für die Rechteerweiterung können in der Ansible-Konfigurationsdatei konfiguriert oder unter Verwendung der Optionen in Ad-hoc-Befehlen definiert werden. Wenn sie unter Verwendung der Optionen in Ad-hoc-Befehlen definiert wurden, haben sie Vorrang vor der Anweisung, die in der Ansible-Konfigurationsdatei konfiguriert wurde. In der folgenden Tabelle werden die Befehlszeilenoptionen für jede Konfigurationsdateianweisung angezeigt.

## Ansible-Befehlszeilenoptionen

Konfigurationsdateianweisungen	Befehlszeilenoption
inventory	-i
remote_user	-u
become	--become, -b
become_method	--become-method
become_user	--become-user
become_ask_pass	--ask-become-pass, -K

Vor der Konfiguration dieser Anweisungen mithilfe der Befehlszeilenoptionen können die aktuell definierten Werte durch Konsultieren der Ausgabe von `ansible --help` bestimmt werden.

```
[user@controlnode ~]$ ansible --help
...output omitted...
-b, --become           run operations with become (nopasswd implied)
--become-method=BECOME_METHOD
                      privilege escalation method to use (default=sudo),
                      valid choices: [ sudo | su | pbrun | pfexec | runas |
                      doas ]
--become-user=BECOME_USER
...output omitted...
-u REMOTE_USER, --user=REMOTE_USER
                      connect as this user (default=None)
```



### Literaturhinweise

Manpage `ansible(1)`

#### Working with Patterns: Ansible-Dokumentation

[https://docs.ansible.com/ansible/2.9/user\\_guide/intro\\_patterns.html](https://docs.ansible.com/ansible/2.9/user_guide/intro_patterns.html)

#### Introduction to Ad-Hoc Commands: Ansible-Dokumentation

[http://docs.ansible.com/ansible/2.9/user\\_guide/intro\\_adhoc.html](http://docs.ansible.com/ansible/2.9/user_guide/intro_adhoc.html)

#### Modulindex: Ansible-Dokumentation

[http://docs.ansible.com/ansible/2.9/modules/modules\\_by\\_category.html](http://docs.ansible.com/ansible/2.9/modules/modules_by_category.html)

#### command - Executes a command on a remote node: Ansible-Dokumentation

[http://docs.ansible.com/ansible/2.9/modules/command\\_module.html](http://docs.ansible.com/ansible/2.9/modules/command_module.html)

#### shell - Execute commands in nodes: Ansible-Dokumentation

[http://docs.ansible.com/ansible/2.9/modules/shell\\_module.html](http://docs.ansible.com/ansible/2.9/modules/shell_module.html)

## ► Angeleitete Übung

# Ausführen von Ad-hoc-Befehlen

In dieser Übung führen Sie Ad-hoc-Befehle auf mehreren verwalteten Hosts aus.

## Ergebnisse

Sie sollten in der Lage sein, Befehle auf verwalteten Hosts auf einer Ad-hoc-Basis und unter Anwendung der Rechteerweiterung auszuführen.

Sie werden mithilfe des Benutzerkontos `devops` Ad-hoc-Befehle auf `workstation` und `servera` ausführen. Dieses Konto hat auf `workstation` und `servera` dieselbe sudo-Konfiguration.

## Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab deploy-adhoc start` aus. Dieses Skript stellt sicher, dass der verwaltete Host `servera` im Netzwerk erreichbar ist. Zudem erstellt das Skript das Arbeitsverzeichnis `/home/student/deploy-adhoc` und füllt es mit den Materialien, die in dieser Übung verwendet werden.

```
[student@workstation ~]$ lab deploy-adhoc start
```

## Anweisungen

- ▶ 1. Legen Sie die sudo-Konfiguration für das `devops`-Konto auf `workstation` und `servera` fest.
  - 1.1. Legen Sie die sudo-Konfiguration für das `devops`-Konto fest, das bei der Erstellung von `workstation` konfiguriert wurde. Wenn Sie zur Eingabe des Passworts für das `student`-Konto aufgefordert werden, geben Sie `student` ein.

```
[student@workstation ~]$ sudo -l -U devops
...output omitted...
User devops may run the following commands on workstation:
(ALL) NOPASSWD: ALL
```

Beachten Sie, dass der Benutzer über uneingeschränkte sudo-Berechtigungen verfügt, jedoch keine Passwortauthentifizierung erforderlich ist.

- 1.2. Legen Sie die sudo-Konfiguration für das `devops`-Konto fest, das bei der Erstellung von `servera` konfiguriert wurde.

```
[student@workstation ~]$ ssh devops@servera.lab.example.com
[devops@servera ~]$ sudo -l
...output omitted...
User devops may run the following commands on servera:
(ALL) NOPASSWD: ALL
[devops@servera ~]$ exit
```

Beachten Sie, dass der Benutzer über uneingeschränkte sudo-Berechtigungen verfügt, jedoch keine Passwortauthentifizierung erforderlich ist.

- 2. Ändern Sie das Verzeichnis in /home/student/deploy-adhoc, und überprüfen Sie den Inhalt der Dateien `ansible.cfg` und `inventory`.

```
[student@workstation ~]$ cd ~/deploy-adhoc
[student@workstation deploy-adhoc]$ cat ansible.cfg
[defaults]
inventory=inventory
[student@workstation deploy-adhoc]$ cat inventory
[control_node]
localhost

[intranetweb]
servera.lab.example.com
```

Die Konfigurationsdatei verwendet die Datei `inventory` im Verzeichnis als das Ansible-Inventar. Beachten Sie, dass Ansible noch nicht für die Verwendung der Rechteerweiterung konfiguriert ist.

- 3. Führen Sie mit der Hostgruppe `all` und dem Modul `ping` einen Ad-hoc-Befehl aus, um sicherzustellen, dass alle verwalteten Hosts Ansible-Module mit Python ausführen können.

```
[student@workstation deploy-adhoc]$ ansible all -m ping
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
localhost | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
```

- 4. Führen Sie mit dem Modul `command` einen Ad-hoc-Befehl auf `workstation` aus, um das Benutzerkonto zu identifizieren, das von Ansible zur Durchführung von Vorgängen auf verwalteten Hosts verwendet wird. Stellen Sie unter Verwendung des Hostmusters `localhost` eine Verbindung mit `workstation` her, um den Ad-hoc-Befehl auszuführen.

## Kapitel 2 | Implementieren eines Ansible-Playbooks

Da die Verbindung lokal hergestellt wird, ist `workstation` gleichzeitig der Kontrollknoten und der verwaltete Host.

```
[student@workstation deploy-adhoc]$ ansible localhost -m command -a 'id'
localhost | CHANGED | rc=0 >>
uid=1000(student) gid=1000(student) groups=1000(student),10(wheel)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Beachten Sie, dass der Ad-hoc-Befehl auf dem verwalteten Host als Benutzer `student` durchgeführt wurde.

- 5. Führen Sie den vorherigen Ad-hoc-Befehl auf `workstation` aus; die Verbindung und der Vorgang müssen jedoch über das Benutzerkonto `devops` mithilfe der Option `-u` vorgenommen werden.

```
[student@workstation deploy-adhoc]$ ansible localhost -m command -a 'id' -u devops
localhost | CHANGED | rc=0 >>
uid=1001(devops) gid=1001(devops) groups=1001(devops)
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Beachten Sie, dass der Ad-hoc-Befehl auf dem verwalteten Host als Benutzer `devops` durchgeführt wurde.

- 6. Führen Sie mithilfe des Moduls `copy` einen Ad-hoc-Befehl auf `workstation` aus, um den Inhalt der Datei `/etc/motd` so zu ändern, dass er aus der Zeichenfolge „Managed by Ansible“ gefolgt von einer neuen Zeile besteht. Führen Sie den Befehl mithilfe des Benutzerkontos `devops` aus, aber verwenden Sie nicht die Option `--become`, um zu `root` zu wechseln. Der Ad-hoc-Befehl sollte aufgrund fehlender Berechtigungen fehlschlagen.

```
[student@workstation deploy-adhoc]$ ansible localhost -m copy \
> -a 'content="Managed by Ansible\n" dest=/etc/motd' -u devops
localhost | FAILED! => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "checksum": "4458b979ede3c332f8f2128385df4ba305e58c27",
    "msg": "Destination /etc not writable"
}
```

Der Ad-hoc-Befehl schlug fehl, weil der Benutzer `devops` keine Schreibberechtigung für die Datei besitzt.

- 7. Führen Sie den Befehl erneut mit Rechteerweiterung aus. Sie könnten die Einstellungen in der Datei `ansible.cfg` korrigieren, aber in diesem Beispiel verwenden Sie einfach geeignete Befehlszeilenoptionen für den Befehl `ansible`.

Führen Sie mithilfe des Moduls `copy` den vorherigen Befehl auf `workstation` aus, um den Inhalt der Datei `/etc/motd` so zu ändern, dass er aus der Zeichenfolge „Managed by Ansible“ gefolgt von einer neuen Zeile besteht. Stellen Sie als Benutzer `devops` die Verbindung zum verwalteten Host her, aber führen Sie den Vorgang als Benutzer `root` mithilfe der Option `--become` durch. Es genügt, die Option `--become` zu verwenden,

da der Standardwert für die Anweisung `become _user` in der Datei `/etc/ansible/ansible.cfg` auf `root` festgelegt ist.

```
[student@workstation deploy-adhoc]$ ansible localhost -m copy \
> -a 'content="Managed by Ansible\n" dest=/etc/motd' -u devops --become
localhost | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": true,
    "checksum": "4458b979ede3c332f8f2128385df4ba305e58c27",
    "dest": "/etc/motd",
    "gid": 0,
    "group": "root",
    "md5sum": "65a4290ee5559756ad04e558b0e0c4e3",
    "mode": "0644",
    "owner": "root",
    "secontext": "system_u:object_r:etc_t:s0",
    "size": 19,
    "src": "/home/devops/.ansible/tmp/ansible-
tmp-1558954193.0260043-131348380629718/source",
    "state": "file",
    "uid": 0
}
```

Beachten Sie, dass der Befehl dieses Mal erfolgreich ist, weil der Ad-hoc-Befehl unter Anwendung der Rechteerweiterung ausgeführt wurde.

- 8. Führen Sie den vorherigen Ad-hoc-Befehl unter Verwendung der Hostgruppe `all` erneut auf allen Hosts aus. Auf diese Weise wird sichergestellt, dass `/etc/motd` sowohl auf `workstation` als auch auf `servera` aus dem Text „Managed by Ansible“ besteht.

```
[student@workstation deploy-adhoc]$ ansible all -m copy \
> -a 'content="Managed by Ansible\n" dest=/etc/motd' -u devops --become
servera.lab.example.com | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": true,
    "checksum": "4458b979ede3c332f8f2128385df4ba305e58c27",
    "dest": "/etc/motd",
    "gid": 0,
    "group": "root",
    "md5sum": "65a4290ee5559756ad04e558b0e0c4e3",
    "mode": "0644",
    "owner": "root",
    "secontext": "system_u:object_r:etc_t:s0",
    "size": 19,
    "src": "/home/devops/.ansible/tmp/ansible-
tmp-1558954250.7893758-136255396678462/source",
    "state": "file",
    "uid": 0
}
localhost | SUCCESS => {
```

## Kapitel 2 | Implementieren eines Ansible-Playbooks

```
"ansible_facts": {  
    "discovered_interpreter_python": "/usr/libexec/platform-python"  
,  
    "changed": false,  
    "checksum": "4458b979ede3c332f8f2128385df4ba305e58c27",  
    "dest": "/etc/motd",  
    "gid": 0,  
    "group": "root",  
    "mode": "0644",  
    "owner": "root",  
    "path": "/etc/motd",  
    "secontext": "system_u:object_r:etc_t:s0",  
    "size": 19,  
    "state": "file",  
    "uid": 0  
}
```

Für localhost sollte SUCCESS und für servera sollte CHANGED angezeigt werden.  
localhost sollte jedoch "changed": false melden, da die Datei sich bereits im richtigen Status befindet. servera sollte umgekehrt "changed": true melden, da die Datei durch den Ad-hoc-Befehl auf den richtigen Status aktualisiert wurde.

- ▶ 9. Führen Sie mithilfe des Moduls command einen Ad-hoc-Befehl aus, um zu verifizieren, dass der Inhalt der Datei cat /etc/motd auf workstation und auf servera erfolgreich geändert wurde. Verwenden Sie die Hostgruppe all und den Benutzer devops, um die verwalteten Hosts anzugeben und die Verbindung zu ihnen herzustellen. Sie benötigen keine Rechteerweiterung, damit dieser Befehl funktioniert.

```
[student@workstation deploy-adhoc]$ ansible all -m command \  
> -a 'cat /etc/motd' -u devops  
servera.lab.example.com | CHANGED | rc=0 >>  
Managed by Ansible  
  
localhost | CHANGED | rc=0 >>  
Managed by Ansible
```

## Beenden

Führen Sie auf workstation das Skript lab deploy-adhoc finish aus, um diese Übung zu bereinigen.

```
[student@workstation ~]$ lab deploy-adhoc finish
```

Hiermit ist die angeleitete Übung beendet.

# Schreiben und Ausführen von Playbooks

---

## Ziele

Am Ende dieses Abschnitts sollten Sie in der Lage sein, ein einfaches Ansible-Playbook zu schreiben und es mit dem Befehl `ansible-playbook` auszuführen.

## Ansible-Playbooks und Ad-hoc-Befehle

Mit Ad-hoc-Befehlen lässt sich eine einzelne, einfache Aufgabe als einmaliger Befehl auf einem Satz von Zielhosts ausführen. Das wahre Leistungsvermögen von Ansible besteht jedoch darin, dass Sie lernen, wie mehrere komplexe Aufgaben mithilfe von Playbooks auf einfache, wiederholbare Weise auf einem Satz von Zielhosts ausgeführt werden können.

Eine *Aufgabe* ist die Anwendung eines Moduls, um eine bestimmte Arbeitseinheit auszuführen. Ein *Play* ist eine Abfolge von Aufgaben, die in einer bestimmten Reihenfolge auf einen oder mehrere aus Ihrem Inventar ausgewählte Hosts angewendet werden. Ein *Playbook* ist eine Textdatei, die eine Liste mit einem oder mehreren Plays enthält, die in einer entsprechenden Reihenfolge ausgeführt werden sollen.

Mit Plays kann ein langer und komplexer Satz manueller administrativer Aufgaben in eine leicht wiederholbare Routine mit vorhersehbaren und erfolgreichen Ergebnissen geändert werden. In einem Playbook können Sie die Abfolge von Aufgaben in einem Play in einer für Benutzer lesbaren und sofort ausführbaren Form speichern. Die Aufgaben selbst dokumentieren durch die Art und Weise, in der sie geschrieben sind, die Schritte, die zur Bereitstellung Ihrer Anwendung oder Infrastruktur notwendig sind.

## Formatieren eines Ansible-Playbooks

Damit Sie das Format eines Playbooks verstehen, betrachten wir nochmals einen Ad-hoc-Befehl, der in einem vorherigen Kapitel verwendet wurde:

```
[student@workstation ~]$ ansible -m user -a "name=newbie uid=4000 state=present" \
> servera.lab.example.com
```

Dieser kann in ein Play mit einzelner Aufgabe umgeschrieben und in einem Playbook gespeichert werden. Das resultierende Playbook wird wie folgt angezeigt:

```
---
- name: Configure important user consistently
  hosts: servera.lab.example.com
  tasks:
    - name: newbie exists with UID 4000
      user:
        name: newbie
        uid: 4000
        state: present
```

## Kapitel 2 | Implementieren eines Ansible-Playbooks

Ein Playbook ist eine im YAML-Format geschriebene Textdatei und wird normalerweise mit der Erweiterung `yml` gespeichert. Das Playbook verwendet Einrückungen mit Leerzeichen, um die Struktur seiner Daten zu signalisieren. YAML stellt keine strengen Anforderungen hinsichtlich der Anzahl der für die Einrückung verwendeten Leerzeichen, aber es gibt zwei Grundregeln.

- Datenelemente auf der gleichen Hierarchieebene (z. B. Elemente in derselben Liste) müssen die gleiche Einrückung aufweisen.
- Elemente, die einem anderen Element untergeordnet sind, müssen stärker eingerückt werden als ihre übergeordneten Elemente.

Zur besseren Lesbarkeit können auch Leerzeilen hinzugefügt werden.



### Wichtig

Für Einrückungen kann nur das Leerzeichen verwendet werden, Tabulatorzeichen sind nicht zulässig.

Wenn Sie den Texteditor `vi` verwenden, können Sie einige Einstellungen anwenden, die das Bearbeiten Ihrer Playbooks möglicherweise erleichtern. Beispielsweise können Sie Ihrer Datei `$HOME/.vimrc` die folgende Zeile hinzufügen, und wenn `vi` erkennt, dass Sie eine YAML-Datei bearbeiten, wird bei Drücken der Tab-Taste eine Einrückung um zwei Leerzeichen vorgenommen, und die nachfolgenden Zeilen werden automatisch eingerückt.

```
autocmd FileType yaml setlocal ai ts=2 sw=2 et
```

Ein Playbook beginnt mit einer Zeile, die aus drei Bindestrichen (---) besteht, welche den Anfang des Dokuments kennzeichnen. Es kann auch mit drei Punkten (...) enden, welche das Ende des Dokuments kennzeichnen, obwohl dies in der Praxis oftmals ausgelassen wird.

Zwischen diesen Kennzeichnungen ist das Playbook als Liste von Plays definiert. Elemente in einer YAML-Liste beginnen mit einem einzelnen Bindestrich, gefolgt von einem Leerzeichen. Eine YAML-Liste könnte folgendermaßen aussehen:

```
- apple
- orange
- grape
```

In vorherigen Playbook beginnt die Zeile nach --- mit einem Bindestrich und startet das erste (und einzige) Play in der Liste von Plays.

Das Play an sich ist eine Sammlung von Schlüssel-/Wert-Paaren. Schlüssel im gleichen Play sollten dieselbe Einrückung aufweisen. Das folgende Beispiel zeigt einen YAML-Ausschnitt mit drei Schlüsseln. Die ersten beiden Schlüssel besitzen einfache Werte. Der dritte besitzt als Wert eine Liste von drei Elementen.

```
name: just an example
hosts: webservers
tasks:
  - first
  - second
  - third
```

## Kapitel 2 | Implementieren eines Ansible-Playbooks

Im ursprünglichen Beispiel-Play sind die drei Schlüssel `name`, `hosts` und `tasks` vorhanden, da diese Schlüssel alle über dieselbe Einrückung verfügen.

Die erste Zeile des Beispiel-Plays beginnt mit einem Bindestrich und einem Leerzeichen (daraus ergibt sich, dass das Play das erste Element einer Liste ist) und danach dem ersten Schlüssel, dem Attribut `name`. Durch den Schlüssel `name` wird eine beliebige Zeichenfolge als Kennzeichnung mit dem Play verknüpft. Damit wird der Zweck des Plays angegeben. Der Schlüsselname ist optional, wird jedoch empfohlen, da er hilfreich für die Dokumentation Ihres Playbooks ist. Dies ist besonders praktisch, wenn ein Playbook mehrere Plays enthält.

```
- name: Configure important user consistently
```

Der zweite Schlüssel in dem Play ist das Attribut `hosts`, das die Hosts angibt, auf denen die Aufgaben des Plays ausgeführt werden. Wie das Argument für den Befehl `ansible` enthält auch das Attribut `hosts` als Wert ein Hostmuster, wie etwa die Namen von verwalteten Hosts oder Gruppen im Inventar.

```
hosts: servera.lab.example.com
```

Der letzte Schlüssel in dem Play ist das Attribut `tasks`, dessen Wert eine Liste von Aufgaben angibt, die für dieses Play ausgeführt werden sollen. Dieses Beispiel enthält eine einzelne Aufgabe, die das Modul `user` mit bestimmten Argumenten ausführt (um sicherzustellen, dass der Benutzer `newbie` vorhanden ist und die UID 4000 besitzt).

```
tasks:
  - name: newbie exists with UID 4000
    user:
      name: newbie
      uid: 4000
      state: present
```

Genau genommen ist das Attribut `tasks` derjenige Teil des Plays, in dem die Aufgaben, die auf den verwalteten Hosts ausgeführt werden sollen, in der entsprechenden Reihenfolge aufgeführt sind. Jede Aufgabe in der Liste ist selbst eine Sammlung von Schlüssel-Wert-Paaren.

In diesem Beispiel enthält die einzige Aufgabe in dem Play zwei Schlüssel:

- `name` ist eine optionale Kennzeichnung, die den Zweck der Aufgabe dokumentiert. Es ist ratsam, Ihre gesamten Aufgaben mit Namen zu versehen. Dies ist hilfreich, um den Zweck der einzelnen Schritte des Automatisierungsprozesses zu dokumentieren.
- `user` ist das Modul, das für diese Aufgabe ausgeführt werden soll. Seine Argumente werden als Sammlung von Schlüssel/Wert-Paaren übergeben, die untergeordnete Elemente des Moduls sind (`name`, `uid` und `state`).

Es folgt ein weiteres Beispiel eines `tasks`-Attributs mit mehreren Aufgaben, die mithilfe des Moduls `service` sicherstellen, dass verschiedene Netzwerkservices für den Start beim Booten konfiguriert werden:

```
tasks:
  - name: web server is enabled
    service:
      name: httpd
      enabled: true
```

```
- name: NTP server is enabled
  service:
    name: chronyd
    enabled: true

- name: Postfix is enabled
  service:
    name: postfix
    enabled: true
```

**Wichtig**

Die Reihenfolge, in der die Plays und Aufgaben in einem Playbook aufgeführt werden, ist wichtig, weil sie von Ansible in derselben Reihenfolge ausgeführt werden.

Die Playbooks, die Sie bisher gesehen haben, sind einfache Beispiele. Im weiteren Verlauf des Kurses werden komplexere Beispiele für die Einsatzmöglichkeiten von Plays und Aufgaben vorgestellt.

## Ausführen von Playbooks

Playbooks können mit dem Befehl `ansible-playbook` ausgeführt werden. Der Befehl wird auf dem Kontrollknoten ausgeführt und der Name des Playbooks, das ausgeführt werden soll, wird als Argument übergeben:

```
[student@workstation ~]$ ansible-playbook site.yml
```

Wenn Sie das Playbook ausführen, wird eine Ausgabe generiert, die das Play und die Aufgaben zeigt, die ausgeführt werden. Die Ausgabe meldet auch die Ergebnisse der einzelnen ausgeführten Aufgaben.

Das folgende Beispiel zeigt den Inhalt eines einfachen Playbooks und danach das Ergebnis seiner Ausführung.

```
[student@workstation playdemo]$ cat webserver.yml
---
- name: play to setup web server
  hosts: servera.lab.example.com
  tasks:
    - name: latest httpd version installed
      yum:
        name: httpd
        state: latest
...output omitted...
[student@workstation playdemo]$ ansible-playbook webserver.yml

PLAY [play to setup web server] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [latest httpd version installed] ****
```

## Kapitel 2 | Implementieren eines Ansible-Playbooks

```
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```

Der Wert des Schlüssels `name` für die einzelnen Plays und Aufgaben wird angezeigt, wenn das Playbook ausgeführt wird. (Die Aufgabe `Gathering Facts` ist eine besondere Aufgabe, die vom Modul `setup` normalerweise automatisch zu Beginn eines Plays ausgeführt wird. Dies wird an späterer Stelle im Kurs behandelt.) Bei Playbooks mit mehreren Plays und Aufgaben sorgt das Festlegen der `name`-Attribute dafür, dass sich der Fortschritt bei der Ausführung eines Playbooks leichter überwachen lässt.

Sie sollten auch feststellen, dass die Aufgabe `latest httpd version installed` für `servera.lab.example.com` den Status `changed` aufweist. Dies bedeutet, dass die Aufgabe auf diesem Host irgendetwas geändert hat, um sicherzustellen, dass dessen Spezifikation erfüllt wurde. In diesem Fall bedeutet es, dass das `httpd`-Paket wahrscheinlich nicht oder nicht in der neuesten Version installiert war.

In der Regel sind Aufgaben in Ansible-Playbooks idempotent und können gefahrlos mehrere Male ausgeführt werden. Wenn die verwalteten Zielhosts bereits den richtigen Status aufweisen, sollten keine Änderungen vorgenommen werden. Nehmen wir z. B. an, dass das Playbook aus dem vorherigen Beispiel erneut ausgeführt wird:

```
[student@workstation playdemo]$ ansible-playbook webserver.yml

PLAY [play to setup web server] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [latest httpd version installed] ****
ok: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=2    changed=0    unreachable=0    failed=0
```

Dieses Mal wurden alle Aufgaben erfolgreich, also mit Status `ok` abgeschlossen und es wurden keine Änderungen gemeldet.

## Steigern der Ausgabeausführlichkeit

Die Standardausgabe des Befehls `ansible-playbook` enthält keine detaillierten Informationen zur Aufgabenausführung. Mit dem Befehl `ansible-playbook -v` werden zusätzliche Informationen mit bis zu vier Gesamtebenen ausgegeben.

### Konfigurieren der Ausgabeausführlichkeit der Playbook-Ausführung

Option	Beschreibung
<code>-v</code>	Die Aufgabenergebnisse werden angezeigt.
<code>-vv</code>	Es werden sowohl die Aufgabenergebnisse als auch die Aufgabenkonfiguration angezeigt.

Option	Beschreibung
-vvv	Schließt Information über die Verbindungen zu verwalteten Hosts ein.
-vvvv	Fügt den Verbindungs-Plugins weitere Optionen für ausführliche Informationen hinzu, einschließlich der Benutzer, mit denen auf den verwalteten Host Skripte ausgeführt werden, und welche Skripte ausgeführt wurden.

## Syntaxverifizierung

Vor der Ausführung eines Playbooks ist es eine bewährte Vorgehensweise, eine Verifizierung vorzunehmen, um die korrekte Syntax des Inhalts zu gewährleisten. Der Befehl `ansible-playbook` bietet eine Option `--syntax-check`, die Sie verwenden können, um die Syntax eines Playbooks zu verifizieren. Das folgende Beispiel zeigt eine erfolgreiche Syntaxverifizierung eines Playbooks.

```
[student@workstation ~]$ ansible-playbook --syntax-check webserver.yml
playbook: webserver.yml
```

Wenn die Syntaxverifizierung fehlschlägt, dann wird ein Syntaxfehler gemeldet. Die Ausgabe umfasst ebenfalls den ungefähren Ort des Syntaxproblems im Playbook. Das folgende Beispiel zeigt die fehlgeschlagene Syntaxverifizierung eines Playbooks, in dem das Leerstellen-Trennzeichen nach dem Attribut `name` für das Play fehlt.

```
[student@workstation ~]$ ansible-playbook --syntax-check webserver.yml
ERROR! Syntax Error while loading YAML.
  mapping values are not allowed in this context

The error appears to have been in ...output omitted... line 3, column 8, but may
be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

  - name:play to setup web server
    hosts: servera.lab.example.com
      ^ here
```

## Ausführung eines Probelaufs

Sie können die Option `-C` verwenden, um einen *Probelauf* der Playbook-Ausführung auszuführen. Hiermit meldet Ansible, welche Änderungen bei der Ausführung des Playbooks eintreten würden, aber tatsächlich werden diese Änderungen an den verwalteten Hosts nie durchgeführt.

Das folgende Beispiel zeigt den Probelauf eines Playbooks mit einer einfachen Aufgabe, bei der die aktuellste Version des Pakets `httpd` auf einem verwalteten Host installiert wird. Beachten Sie, dass der Probelauf meldet, dass die Aufgabe eine Änderung am verwalteten Host verursachen würde.

```
[student@workstation ~]$ ansible-playbook -C webserver.yml

PLAY [play to setup web server] ****

TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [latest httpd version installed] ****
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```



### Literaturhinweise

Manpage `ansible-playbook(1)`

#### **Intro to Playbooks – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_intro.html](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_intro.html)

#### **Playbooks – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks.html](https://docs.ansible.com/ansible/2.9/user_guide/playbooks.html)

#### **Check Mode ("Dry Run") – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_checkmode.html](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_checkmode.html)

## ► Angeleitete Übung

# Schreiben und Ausführen von Playbooks

In dieser Übung schreiben Sie ein Ansible-Playbook und führen es aus.

## Ergebnisse

Sie sollten in der Lage sein, mithilfe von einfacher YAML-Syntax und der Ansible-Playbook-Struktur ein Playbook zu schreiben und es mit dem Befehl `ansible-playbook` erfolgreich auszuführen.

## Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab playbook-basic start` aus. Diese Funktion stellt sicher, dass die verwalteten Hosts `serverc.lab.example.com` und `serverd.lab.example.com` im Netzwerk erreichbar sind. Dies gewährleistet auch die Installation der korrekten Ansible-Konfigurationsdatei und Inventardatei auf dem Kontrollknoten.

```
[student@workstation ~]$ lab playbook-basic start
```

## Anweisungen

Für diese Übung wurde das Arbeitsverzeichnis `/home/student/playbook-basic` auf `workstation` erstellt. Dieses Verzeichnis wurde bereits mit der Konfigurationsdatei `ansible.cfg` und auch mit einer Inventardatei `inventory` gefüllt, in der die Gruppe `web` definiert ist, die die beiden oben angeführten verwalteten Hosts als Mitglieder enthält.

Verwenden Sie in diesem Verzeichnis einen Texteditor, um ein Playbook mit dem Namen `site.yml` zu erstellen. Dieses Playbook enthält ein einzelnes Play, das Mitglieder der Hostgruppe `web` als Ziel verwenden sollte. Das Playbook sollte mithilfe von Aufgaben sicherstellen, dass auf den verwalteten Hosts die folgenden Bedingungen erfüllt sind:

- Das `httpd`-Paket ist vorhanden (mithilfe des Moduls `yum`).
- Die lokale Datei `files/index.html` wird auf jedem verwalteten Host nach `/var/www/html/index.html` kopiert (mithilfe des Moduls `copy`).
- Der Service `httpd` wird mithilfe des Moduls `service` gestartet und aktiviert.

Es ist hilfreich, den Befehl `ansible-doc` zu verwenden, um die Keywords nachzuvollziehen, die für die einzelnen Module benötigt werden.

Verifizieren Sie nach der Erstellung des Playbooks dessen Syntax und führen Sie dann mithilfe von `ansible-playbook` das Playbook aus, um die Konfiguration zu implementieren.

- 1. Wechseln Sie in das Verzeichnis `/home/student/playbook-basic`.

```
[student@workstation ~]$ cd ~/playbook-basic  
[student@workstation playbook-basic]$
```

- 2. Verwenden Sie einen Texteditor, um ein neues Playbook mit dem Namen `/home/student/playbook-basic/site.yml` zu erstellen. Schreiben Sie ein Play, das auf die Hosts in der Hostgruppe `web` abzielt.

- 2.1. Erstellen und öffnen Sie `~/playbook-basic/site.yml`. Die erste Zeile der Datei sollte aus drei Bindestrichen bestehen, um den Anfang des Playbooks zu kennzeichnen.

```
---
```

- 2.2. In der nächsten Zeile beginnt das Play. Es muss mit einem Bindestrich und einem Leerzeichen vor dem ersten Keyword im Play beginnen. Benennen Sie das Play mithilfe des `name`-Keywords mit einer beliebigen Zeichenfolge, die den Zweck des Plays dokumentiert.

```
---
```

```
- name: Install and start Apache HTTPD
```

- 2.3. Fügen Sie das Keyword-/Wert-Paar `hosts` hinzu, um anzugeben, dass das Play auf Hosts der Hostgruppe `web` des Inventars ausgeführt wird. Stellen Sie sicher, dass das Keyword `hosts` um zwei Leerzeichen eingerückt ist, damit es am Keyword `name` der vorherigen Zeile ausgerichtet ist.

Die vollständige Datei `site.yml` sollte nun folgendermaßen aussehen:

```
---
```

```
- name: Install and start Apache HTTPD  
hosts: web
```

- 3. Setzen Sie die Bearbeitung der Datei `/home/student/playbook-basic/site.yml` fort, und fügen Sie ein `tasks`-Keyword und drei Aufgaben für Ihr Play hinzu, die in den Anweisungen spezifiziert wurden.

- 3.1. Fügen Sie ein `tasks`-Keyword hinzu, die um zwei Leerzeichen eingerückt ist (in einer Linie mit dem `hosts`-Keyword), um die Liste der Aufgaben zu beginnen. Ihre Datei sollte nun folgendermaßen aussehen:

```
---
```

```
- name: Install and start Apache HTTPD  
hosts: web  
tasks:
```

- 3.2. Fügen Sie die erste Aufgabe hinzu. Rücken Sie mit vier Leerzeichen ein, beginnen Sie die Aufgabe mit einem Bindestrich und einem Leerzeichen und geben Sie ihr dann einen Namen, wie etwa `httpd package is present`. Verwenden Sie das `yum`-Modul für diese Aufgabe. Rücken Sie die Modul-Keywords um weitere zwei Leerzeichen ein. Legen Sie den Paketnamen auf `httpd` und den Paketzustand auf `present` fest. Die Aufgabe sollte folgendermaßen aussehen:

```
- name: httpd package is present
yum:
  name: httpd
  state: present
```

- 3.3. Fügen Sie die zweite Aufgabe hinzu. Verwenden Sie dasselbe Format wie in der vorherigen Aufgabe und geben Sie der Aufgabe einen Namen, wie etwa `correct index.html is present`. Verwenden Sie das Modul `copy`. Die Keywords des Moduls sollten den Schlüssel `src` auf `files/index.html` und den Schlüssel `dest` auf `/var/www/html/index.html` festlegen. Die Aufgabe sollte folgendermaßen aussehen:

```
- name: correct index.html is present
copy:
  src: files/index.html
  dest: /var/www/html/index.html
```

- 3.4. Fügen Sie die dritte Aufgabe hinzu, die den Service `httpd` startet und aktiviert. Verwenden Sie dasselbe Format wie in der beiden vorherigen Aufgaben und geben Sie der neuen Aufgabe einen Namen, wie etwa `httpd is started`. Verwenden Sie das `service`-Modul für diese Aufgabe. Legen Sie den Schlüssel `name` des Services auf `httpd`, den Schlüssel `state` auf `started` und den Schlüssel `enabled` auf `true` fest. Die Aufgabe sollte folgendermaßen aussehen:

```
- name: httpd is started
service:
  name: httpd
  state: started
  enabled: true
```

- 3.5. Ihr vollständiges Ansible-Playbook `site.yml` sollte mit dem folgenden Beispiel übereinstimmen. Stellen Sie sicher, dass sämtliche Einrückungen der Keywords in Ihrem Play, die Liste von Aufgaben sowie die Keywords in den einzelnen Aufgaben korrekt sind.

```
---
- name: Install and start Apache HTTPD
  hosts: web
  tasks:
    - name: httpd package is present
      yum:
        name: httpd
        state: present

    - name: correct index.html is present
      copy:
        src: files/index.html
        dest: /var/www/html/index.html

    - name: httpd is started
      service:
```

```
name: httpd
state: started
enabled: true
```

Speichern Sie die Datei und beenden Sie den Texteditor.

- 4. Führen Sie vor dem Ausführen Ihres Playbooks den Befehl `ansible-playbook --syntax-check site.yml` aus, um zu verifizieren, dass die zugehörige Syntax korrekt ist. Wenn Fehler gemeldet werden, korrigieren Sie sie, bevor Sie mit dem nächsten Schritt fortfahren. Es sollte ungefähr folgende Ausgabe angezeigt werden:

```
[student@workstation playbook-basic]$ ansible-playbook --syntax-check site.yml

playbook: site.yml
```

- 5. Führen Sie Ihr Playbook aus. Überprüfen Sie die generierte Ausgabe, um den erfolgreichen Abschluss aller Aufgaben zu gewährleisten.

```
[student@workstation playbook-basic]$ ansible-playbook site.yml

PLAY [Install and start Apache HTTPD] ****

TASK [Gathering Facts] ****
ok: [serverd.lab.example.com]
ok: [serverc.lab.example.com]

TASK [httpd package is present] ****
changed: [serverd.lab.example.com]
changed: [serverc.lab.example.com]

TASK [correct index.html is present] ****
changed: [serverd.lab.example.com]
changed: [serverc.lab.example.com]

TASK [httpd is started] ****
changed: [serverd.lab.example.com]
changed: [serverc.lab.example.com]

PLAY RECAP ****
serverc.lab.example.com    : ok=4      changed=3      unreachable=0      failed=0
serverd.lab.example.com    : ok=4      changed=3      unreachable=0      failed=0
```

- 6. Wenn alles gut verläuft, sollten Sie das Playbook ein zweites Mal ausführen können und feststellen, dass alle Aufgaben ohne Änderungen an den verwalteten Hosts abgeschlossen wurden.

```
[student@workstation playbook-basic]$ ansible-playbook site.yml

PLAY [Install and start Apache HTTPD] ****

TASK [Gathering Facts] ****
ok: [serverd.lab.example.com]
ok: [serverc.lab.example.com]
```

```
TASK [httpd package is present] ****
ok: [serverd.lab.example.com]
ok: [serverc.lab.example.com]

TASK [correct index.html is present] ****
ok: [serverc.lab.example.com]
ok: [serverd.lab.example.com]

TASK [httpd is started] ****
ok: [serverd.lab.example.com]
ok: [serverc.lab.example.com]

PLAY RECAP ****
serverc.lab.example.com      : ok=4    changed=0    unreachable=0   failed=0
serverd.lab.example.com      : ok=4    changed=0    unreachable=0   failed=0
```

- 7. Führen Sie den Befehl `curl` aus, um zu verifizieren dass `serverc` und `serverd` als HTTPD-Server konfiguriert sind.

```
[student@workstation playbook-basic]$ curl serverc.lab.example.com
This is a test page.
[student@workstation playbook-basic]$ curl serverd.lab.example.com
This is a test page.
```

## Beenden

Führen Sie auf `workstation` das Skript `lab playbook-basic finish` aus, um die in dieser Übung erstellten Ressourcen zu bereinigen.

```
[student@workstation ~]$ lab playbook-basic finish
```

Hiermit ist die angeleitete Übung beendet.

# Implementieren mehrerer Plays

## Ziele

Am Ende dieses Abschnitts sollten Sie in der Lage sein, ein Playbook zu schreiben, das mehrere Plays und die Rechteerweiterung auf Play-Basis verwendet, und `ansible-doc` effektiv zu verwenden, um zu lernen, wie Sie neue Module verwenden, um Aufgaben für ein Play zu implementieren.

## Schreiben mehrerer Plays

Ein Playbook ist eine YAML-Datei, die eine Liste mit einem oder mehreren Plays enthält. Denken Sie daran, dass es sich bei einem einzelnen Play um eine geordnete Liste von Aufgaben handelt, die auf Hosts ausgeführt werden sollen, die aus dem Inventar ausgewählt werden. Wenn ein Playbook mehrere Plays enthält, kann daher jedes Play seine Aufgaben auf einen eigenen Satz von Hosts anwenden.

Dies kann sehr nützlich sein, wenn eine komplexe Bereitstellung orchestriert wird, die verschiedene Aufgaben auf unterschiedlichen Hosts erfordert kann. Sie können ein Playbook schreiben, mit dem ein Play auf einer Reihe von Hosts ausgeführt wird und danach ein weiteres Play auf einer anderen Reihe von Hosts.

Ein Playbook, das mehrere Plays enthält, lässt sich sehr einfach schreiben. Jedes Play in dem Playbook wird dort als Listenelement auf oberster Ebene erstellt. Jedes Play ist ein Listenelement, das die üblichen Play-Keywords enthält.

Das folgende Beispiel zeigt ein einfaches Playbook mit zwei Plays. Das erste Play wird auf `web.example.com` und das zweite Play auf `database.example.com` ausgeführt.

```
---
# This is a simple playbook with two plays

- name: first play
  hosts: web.example.com
  tasks:
    - name: first task
      yum:
        name: httpd
        status: present

    - name: second task
      service:
        name: httpd
        enabled: true

- name: second play
  hosts: database.example.com
  tasks:
    - name: first task
```

```
service:  
  name: mariadb  
  enabled: true
```

## Remote-Benutzer und Rechteerweiterung in Plays

Plays können auch andere Remote-Benutzer oder Einstellungen für die Rechteerweiterung verwenden als diejenigen, die mit den Standardwerten in der Konfigurationsdatei festgelegt werden. Diese werden im Play selbst auf derselben Ebene wie die Keywords `hosts` oder `tasks` festgelegt.

### Benutzerattribute

Aufgaben in Playbooks werden normalerweise über eine Netzwerkverbindung mit den verwalteten Hosts ausgeführt. Wie bei Ad-hoc-Befehlen hängt das für die Ausführung von Aufgaben verwendete Benutzerkonto von verschiedenen Keywords in der Ansible-Konfigurationsdatei `/etc/ansible/ansible.cfg` ab. Der Benutzer, der die Aufgabe ausführt, kann durch das Keyword `remote_user` definiert werden. Wenn jedoch die Rechteerweiterung aktiviert ist, können auch andere Keywords, beispielsweise `become_user`, einen Einfluss haben.

Wenn der in der Ansible-Konfiguration definierte Remote-Benutzer für die Aufgabenausführung nicht geeignet ist, kann dies mithilfe des Keywords `remote_user` innerhalb eines Plays überschrieben werden.

```
remote_user: remoteuser
```

### Attribute für die Rechteerweiterung

Es sind noch weitere Keywords verfügbar, um Parameter für die Rechteerweiterung innerhalb eines Playbooks zu definieren. Mithilfe des booleschen Keywords `become` kann die Rechteerweiterung aktiviert oder deaktiviert werden, unabhängig davon, wie sie in der Ansible-Konfigurationsdatei definiert ist. Der Parameter kann `yes` oder `true` enthalten, damit die Rechteerweiterung aktiviert wird, bzw. `no` oder `false`, um sie zu deaktivieren.

```
become: true
```

Ist die Rechteerweiterung aktiviert, kann das Keyword `become_method` verwendet werden, um die Methode der Rechteerweiterung innerhalb eines bestimmten Plays zu definieren. Im unten stehenden Beispiel ist angegeben, dass `sudo` für die Rechteerweiterung verwendet werden soll.

```
become_method: sudo
```

Zusätzlich kann mit dem Keyword `become_user` und aktivierter Rechteerweiterung das Benutzerkonto definiert werden, das für die Rechteerweiterung innerhalb eines bestimmten Plays verwendet werden soll.

```
become_user: privileged_user
```

Das folgende Beispiel veranschaulicht die Verwendung dieser Keywords in einem Play:

```
- name: /etc/hosts is up to date
  hosts: datacenter-west
  remote_user: automation
  become: yes

  tasks:
    - name: server.example.com in /etc/hosts
      lineinfile:
        path: /etc/hosts
        line: '192.0.2.42 server.example.com server'
        state: present
```

## Suche nach Modulen für Aufgaben

### Moduldokumentation

Die große Anzahl der Modulpakete in Ansible liefert Administratoren viele Tools für gängige administrative Aufgaben. An früherer Stelle in diesem Kurs sprachen wir über die Website für die Ansible-Dokumentation unter <http://docs.ansible.com>. Der *Modulindex* auf der Website bietet eine einfache Möglichkeit, die Liste der im Lieferumfang von Ansible enthaltenen Module zu durchsuchen. Module für Benutzer- und Serviceverwaltung finden Sie z. B. unter *Systems Modules* und Module für Datenbankadministration unter *Database Modules*.

Für jedes Modul bietet die Ansible-Dokumentation eine Zusammenfassung der Funktionen und Anweisungen für jede einzelne Funktion mit Optionen für das Modul. Die Dokumentation enthält auch nützliche Beispiele, die zeigen, wie die einzelnen Module verwendet und ihre Keywords in einer Aufgabe festgelegt werden können.

Sie haben bereits mit dem Befehl `ansible-doc` gearbeitet, um nach Informationen über Module zu suchen, die auf dem lokalen System installiert sind. Zur Wiederholung führen Sie den Befehl `ansible-doc -l` aus, um eine Liste der auf dem Kontrollknoten verfügbaren Module anzuzeigen. Dieser Befehl zeigt eine Liste von Modulnamen mit einer Zusammenfassung der jeweiligen Funktionen an.

```
[student@workstation modules]$ ansible-doc -l
a10_server           Manage A10 Networks ... devices' server object.
a10_server_axapi3    Manage A10 Networks ... devices
a10_service_group    Manage A10 Networks ... devices' service groups.
a10_virtual_server   Manage A10 Networks ... devices' virtual servers.
...output omitted...
zfs_facts             Gather facts about ZFS datasets.
znode                 Create, ... and update znodes using ZooKeeper
zpool_facts           Gather facts about ZFS pools.
zypper                Manage packages on SUSE and openSUSE
zypper_repository     Add and remove Zypper repositories
```

Verwenden Sie den Befehl `ansible-doc [module name]` zum Anzeigen einer detaillierten Dokumentation für ein Modul. Wie bei der Dokumentations-Website von Ansible bietet der Befehl eine Zusammenfassung der Modulfunktion, Details der verschiedenen Optionen sowie Beispiele. Das folgende Beispiel zeigt die Dokumentation für das Modul `yum`.

```
[student@workstation modules]$ ansible-doc yum
> YUM      (/usr/lib/python3.6/site-packages/ansible/modules/packaging/os/yum.py)

    Installs, upgrade, downgrades, removes, and lists packages and groups with
    the `yum` package manager. This module only works on Python 2. If you require
    Python
        3 support see the [dnf] module.

    * This module is maintained by The Ansible Core Team
    * note: This module has a corresponding action plugin.

OPTIONS (= is mandatory):

- allow_downgrade
    Specify if the named package and version is allowed to downgrade a maybe
    already installed higher version of that package. Note that setting
        allow_downgrade=True can make this module behave in a non-idempotent way.
    The task could end up with a set of packages that does not match the complete
    list of
        specified packages to install (because dependencies between the downgraded
    package and others can cause changes to the packages which were in the earlier
        transaction).
    [Default: no]
    type: bool
    version_added: 2.4

- autoremove
    If `yes`, removes all "leaf" packages from the system that were originally
    installed as dependencies of user-installed packages but which are no longer
    required
        by any such package. Should be used alone or when state is `absent`
    NOTE: This feature requires yum >= 3.4.3 (RHEL/CentOS 7+)
    [Default: no]
    type: bool
    version_added: 2.7

- bugfix
    If set to `yes`, and `state=latest` then only installs updates that have
    been marked bugfix related.
    [Default: no]
    version_added: 2.6

- conf_file
    The remote yum configuration file to use for the transaction.
    [Default: (null)]
    version_added: 0.6

- disable_excludes
    Disable the excludes defined in YUM config files.
    If set to `all`, disables all excludes.
    If set to `main`, disable excludes defined in [main] in yum.conf.
    If set to `repoid`, disable excludes defined for given repo id.
    [Default: (null)]
    version_added: 2.7
```

```
- disable_gpg_check
    Whether to disable the GPG checking of signatures of packages being
    installed. Has an effect only if state is `present' or `latest'.
    [Default: no]
    type: bool
    version_added: 1.2

- disable_plugin
    `Plugin' name to disable for the install/update operation. The disabled
    plugins will not persist beyond the transaction.
    [Default: (null)]
    version_added: 2.5

- disablerepo
    `Repopid' of repositories to disable for the install/update operation.
    These repos will not persist beyond the transaction. When specifying multiple
    repos,
        separate them with a `",",'.
    As of Ansible 2.7, this can alternatively be a list instead of `",,"'
    separated string
    [Default: (null)]
```

Der Befehl `ansible-doc` bietet zudem die Option `-s`, die eine Beispielausgabe generiert, welche als Modell für die Verwendung eines bestimmten Moduls in einem Playbook fungieren kann. Diese Ausgabe kann als Startvorlage dienen, die in ein Playbook integriert werden kann, um das Modul für die Ausführung von Aufgaben zu implementieren. Kommentare sind in der Ausgabe enthalten, um die Administratoren an die Verwendung jeder Option zu erinnern. Das folgende Beispiel zeigt diese Ausgabe für das Modul `yum`.

```
[student@workstation ~]$ ansible-doc -s yum
- name: Manages packages with the `yum' package manager
  yum:
    allow_downgrade:      # Specify if the named package ...
    autoremove:           # If `yes', removes all "leaf" packages ...
    bugfix:               # If set to `yes', ...
    conf_file:            # The remote yum configuration file ...
    disable_excludes:     # Disable the excludes ...
    disable_gpg_check:   # Whether to disable the GPG ...
    disable_plugin:       # `Plugin' name to disable ...
    disablerepo:          # `Repopid' of repositories ...
    download_only:        # Only download the packages, ...
    enable_plugin:         # `Plugin' name to enable ...
    enablerepo:           # `Repopid' of repositories to enable ...
    exclude:              # Package name(s) to exclude ...
    installroot:          # Specifies an alternative installroot, ...
    list:                 # Package name to run ...
    name:                 # A package name or package specifier ...
    releasever:           # Specifies an alternative release ...
    security:             # If set to `yes', ...
    skip_broken:          # Skip packages with ...
    state:                # Whether to install ... or remove ... a package.
    update_cache:          # Force yum to check if cache ...
```

```
update_only:          # When using latest, only update ...
use_backend:         # This module supports `yum' ...
validate_certs:      # This only applies if using a https url ...
```

## Modulwartung

Im Lieferumfang von Ansible sind zahlreiche Module enthalten, die für viele Aufgaben verwendet werden können. Die Upstream-Community ist sehr aktiv und diese Module können sich in verschiedenen Entwicklungsphasen befinden. In der `ansible-doc`-Dokumentation für das jeweilige Modul soll angegeben werden, wer in der Ansible-Upstream-Community für die Verwaltung dieses Moduls sorgt und in welchem Entwicklungsstatus es sich befindet. Darauf wird im Abschnitt **METADATA** am Ende der Ausgabe von `ansible-doc` für dieses Modul hingewiesen.

In dem Feld `status` wird der Entwicklungsstatus des Moduls erfasst:

- **stableinterface**: Die Keywords des Moduls sind stabil, und es werden alle Anstrengungen unternommen, um Keywords nicht zu entfernen oder ihre Bedeutung zu ändern.
- **preview**: Das Modul befindet sich in der Technologievorschau. Es könnte instabil sein, seine Keywords könnten sich ändern oder es könnte Libraries oder Webservices benötigen, die ihrerseits von inkompatiblen Änderungen betroffen sind.
- **deprecated**: Das Modul ist veraltet und wird in einigen zukünftigen Versionen nicht mehr verfügbar sein.
- **removed**: Das Modul wurde aus der Version entfernt, aber für Dokumentationszwecke ist ein Stub vorhanden, der Benutzer früherer Versionen bei der Migration auf neue Module unterstützt.



### Anmerkung

Der Status `stableinterface` zeigt lediglich an, dass die Schnittstelle des Moduls stabil ist, die Codequalität des Moduls wird jedoch nicht bewertet.

In dem Feld `supported_by` wird erfasst, wer in der Ansible-Upstream-Community das Modul pflegt. Mögliche Werte sind:

- **core**: Von Ansible-Upstream-Kernentwicklern gepflegt und immer in Ansible enthalten.
- **curated**: Die Module werden von Partnern oder Unternehmen in der Community übermittelt und gepflegt. Die Personen, die diese Module pflegen, müssen gemeldete Probleme im Auge behalten und alle Anfragen zum Modul übernehmen. Die Ansible-Kernentwickler prüfen vorgeschlagene Änderungen an „`curated`“-Modulen, nachdem die Maintainer der Community die Änderungen genehmigt haben. Die Kern-Projektbeteiligten stellen außerdem sicher, dass alle Probleme mit diesen Modulen behoben werden, die aufgrund von Änderungen an der Ansible-Engine auftreten. Diese Module sind derzeit in Ansible enthalten, könnten aber irgendwann in der Zukunft in separaten Paketen angeboten werden.
- **community**: Module, die nicht von den wichtigsten Upstream-Entwicklern, Partnern oder Unternehmen unterstützt werden, sondern vollständig von der allgemeinen Open-Source-Community gepflegt werden. Module in dieser Kategorie können weiterhin vollständig genutzt werden, aber die Reaktionszeit bei Problemen ist von der Community abhängig. Diese Module sind ebenfalls derzeit in Ansible enthalten, werden aber irgendwann in der Zukunft in separaten Paketen angeboten.

Die Ansible-Upstream-Community verfügt für Ansible und die dazugehörigen integrierten Module über einen Bugtracker unter <https://github.com/ansible/ansible/issues>.

Manchmal ist kein Modul vorhanden, dass den von Ihnen gewünschten Zweck erfüllt. Als Endbenutzer können Sie auch eigene private Module schreiben oder Module von Dritten beziehen. Ansible sucht an dem Speicherort nach benutzerdefinierten Modulen, der von der Umgebungsvariable `ANSIBLE_LIBRARY` oder, falls diese nicht angegeben ist, von einem `library`-Keyword in der aktuellen Ansible-Konfigurationsdatei festgelegt wird. Ansible sucht auch im Verzeichnis `./library` für das aktuell ausgeführte Playbook nach benutzerdefinierten Modulen.

```
library = /usr/share/my_modules
```

Informationen zum Erstellen von Modulen würden über den Umfang dieses Kurses hinausgehen. Die Dokumentation zu diesem Thema finden Sie unter [https://docs.ansible.com/ansible/2.9/dev\\_guide/developing\\_modules.html](https://docs.ansible.com/ansible/2.9/dev_guide/developing_modules.html).



### Wichtig

Führen Sie den Befehl `ansible-doc` aus, um mehr über die Verwendung von Modulen für Ihre Aufgaben zu erfahren.

Versuchen Sie, die Module `command`, `shell` und `raw` in Playbooks möglichst zu vermeiden, auch wenn sie scheinbar leicht zu verwenden sind. Diese Module enthalten willkürliche Befehle und daher ist es sehr einfach, nicht-idempotente Playbooks mit diesen Modulen zu erstellen.

Die folgende Aufgabe mit dem Modul `shell` ist z. B. nicht idempotent. Jedes Mal, wenn das Play ausgeführt wird, wird die Datei `/etc/resolv.conf` umgeschrieben, selbst wenn sie bereits aus der Zeile `nameserver 192.0.2.1` besteht.

```
- name: Non-idempotent approach with shell module
  shell: echo "nameserver 192.0.2.1" > /etc/resolv.conf
```

Es gibt mehrere Möglichkeiten, das Modul `shell` auf idempotente Weise zu verwenden, um Aufgaben zu schreiben. Manchmal ist der beste Ansatz, diese Änderungen vorzunehmen und `shell` einzusetzen. Eine schnellere Lösung könnte darin bestehen, `ansible-doc` zur Erkennung des Moduls `copy` zu verwenden, um damit den gewünschten Effekt zu erzielen.

Im folgenden Beispiel wird die Datei `/etc/resolv.conf` nicht umgeschrieben, wenn sie bereits den richtigen Inhalt aufweist:

```
- name: Idempotent approach with copy module
  copy:
    dest: /etc/resolv.conf
    content: "nameserver 192.0.2.1\n"
```

Das Modul `copy` kann testen, ob der Status bereits erreicht wurde, und in diesem Fall keine Änderungen vornehmen. Das Modul `shell` bietet viel Flexibilität, erfordert aber auch eine höhere Aufmerksamkeit, um eine idempotente Ausführung zu gewährleisten.

Idempotente Playbooks können wiederholt ausgeführt werden, um einen bestimmten Status von Systemen zu gewährleisten, ohne diese Systeme zu unterbrechen, wenn sie bereits diesen Status aufweisen.

## Varianten der Playbook-Syntax

Im letzten Teil dieses Kapitels betrachten wir einige Varianten der YAML- oder Ansible-Playbook-Syntax, denen Sie möglicherweise begegnen.

### YAML-Kommentare

Kommentare können auch zur besseren Lesbarkeit hinzugefügt werden. In YAML ist alles rechts neben dem Nummern- oder Rautensymbol (#) ein Kommentar. Wenn es Inhalt links vom Kommentar gibt, stellen Sie dem Nummersymbol eine Leerstelle voran.

```
# This is a YAML comment  
  
some data # This is also a YAML comment
```

### YAML-Zeichenfolgen

Zeichenfolgen in YAML müssen normalerweise nicht in Anführungszeichen gesetzt werden, selbst wenn Leerzeichen enthalten sind. Sie können Zeichenfolgen entweder in doppelte oder einfache Anführungszeichen gesetzt werden.

```
this is a string  
  
'this is another string'  
  
"this is yet another a string"
```

Es gibt zwei Möglichkeiten, mehrzeilige Zeichenfolgen zu erstellen. Sie können das vertikale Zeichen (|) verwenden, um anzugeben, dass die Zeilenumbrüche in der Zeichenfolge beibehalten werden sollen.

```
include_newlines: |  
    Example Company  
    123 Main Street  
    Atlanta, GA 30303
```

Sie können auch mit dem Größer-als-Zeichen (>) mehrzeilige Zeichenfolgen erstellen, um anzugeben, dass Zeilenumbrüche in Leerzeichen umgewandelt und führende Leerzeichen in den Zeilen entfernt werden müssen. Diese Methode wird oft genutzt, um lange Zeichenfolgen an Leerzeichen zu trennen, da Mehrzeiler die Lesbarkeit erhöhen.

```
fold_newlines: >  
    This is an example  
    of a long string,  
    that will become  
    a single sentence once folded.
```

## YAML-Dictionaries

Sie haben Sammlungen von Schlüssel/Wert-Paaren gesehen, die wie folgt als eingerückter Block geschrieben sind:

```
name: svcrole
svcservice: httpd
svcport: 80
```

Dictionaries können auch im Blockformat in einer Zeile geschrieben werden, die wie folgt in geschwungene Klammern gesetzt ist:

```
{name: svcrole, svcservice: httpd, svcport: 80}
```

In den meisten Fällen sollte das einzeilige Blockformat vermieden werden, da es schwieriger zu lesen ist. Es gibt jedoch mindestens eine Situation, in der es häufiger verwendet wird. Die Verwendung von *Rollen* wird später in diesem Kurs behandelt. Wenn ein Playbook eine Liste von Rollen enthält, wird diese Syntax häufiger verwendet, um Rollen, die in einem Play einbezogen sind, und Variablen, die an eine Rolle übergeben werden, leichter unterscheiden zu können.

## YAML-Listen

Sie haben auch Listen gesehen, die mit der normalen Syntax mit einzelnen Bindestrichen geschrieben sind:

```
hosts:
  - servera
  - serverb
  - serverc
```

Listen können ebenfalls in einem einzeiligen Format vorliegen, das in eckige Klammern eingeschlossen ist und folgendermaßen aussieht:

```
hosts: [servera, serverb, serverc]
```

Sie sollten diese Syntax vermeiden, da sie normalerweise schwieriger zu lesen ist.

## Veraltete Playbook-Schlüssel/Wert-Kurzsyntax

Manche Playbooks verwenden möglicherweise eine ältere Methode mit Kurzsyntax, um zur Definition von Aufgaben die Schlüssel/Wert-Paare für das Modul in die gleiche Zeile wie den Modulnamen zu platzieren. Es könnte z. B. folgende Syntax verwendet werden:

```
tasks:
  - name: shorthand form
    service: name=httpd enabled=true state=started
```

Normalerweise würden Sie dieselbe Aufgabe folgendermaßen schreiben:

```
tasks:  
  - name: normal form  
    service:  
      name: httpd  
      enabled: true  
      state: started
```

Sie sollten im Allgemeinen die Kurzform vermeiden und die normale Form verwenden.

Die normale Form umfasst mehr Zeilen, aber es ist einfacher, mit dieser Form zu arbeiten. Die Keywords der Aufgabe sind vertikal gestapelt und lassen sich leichter unterscheiden. Ihre Augen können das Play von oben nach unten durchlaufen, bei weniger Bewegung in der Waagerechten. Außerdem ist die normale Syntax native YAML, die Kurzform aber nicht. In modernen Texteditoren können Sie von Tools zur Hervorhebung der Syntax effektiver unterstützt werden, wenn Sie das normale Format statt des Kurzformats verwenden.

Sie könnten dieser Syntax in der Dokumentation und älteren Playbooks anderer Benutzer begegnen, und die Syntax funktioniert nach wie vor.



### Literaturhinweise

Manpages `ansible-playbook(1)` und `ansible-doc(1)`

#### **Intro to Playbooks – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_intro.html](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_intro.html)

#### **Playbooks – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks.html](https://docs.ansible.com/ansible/2.9/user_guide/playbooks.html)

#### **Developing Modules – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/dev\\_guide/developing\\_modules.html](https://docs.ansible.com/ansible/2.9/dev_guide/developing_modules.html)

#### **Module Support – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/user\\_guide/modules\\_support.html](https://docs.ansible.com/ansible/2.9/user_guide/modules_support.html)

#### **YAML-Syntax – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/reference\\_appendices/YAMLSyntax.html](https://docs.ansible.com/ansible/2.9/reference_appendices/YAMLSyntax.html)

## ► Angeleitete Übung

# Implementieren mehrerer Plays

In dieser Übung erstellen Sie ein Playbook, das mehrere Plays enthält, und verwenden es dann, um Konfigurationsaufgaben auf verwalteten Hosts auszuführen.

## Ergebnisse

Sie sollten in der Lage sein, ein Playbook zu erstellen und auszuführen, um eine Konfiguration zu verwalten und Administrationsaufgaben für einen verwalteten Host auszuführen.

## Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab playbook-multi start` aus. Diese Funktion stellt sicher, dass der verwaltete Host `servera.lab.example.com` im Netzwerk erreichbar ist. Dies gewährleistet auch die Installation der korrekten Ansible-Konfigurationsdatei und Inventardatei auf dem Kontrollknoten.

```
[student@workstation ~]$ lab playbook-multi start
```

## Anweisungen

- 1. Auf `workstation` wurde das Arbeitsverzeichnis `/home/student/playbook-multi` für das Ansible-Projekt erstellt. Das Verzeichnis wurde bereits mit der Konfigurationsdatei `ansible.cfg` und der Inventardatei `inventory` gefüllt. In der Inventardatei ist der verwaltete Host `servera.lab.example.com` bereits definiert. Erstellen Sie das neue Playbook `/home/student/playbook-multi/intranet.yml`, und fügen Sie die Zeilen hinzu, die für den Start des ersten Plays benötigt werden. Es sollte den verwalteten Host `servera.lab.example.com` als Ziel verwenden und die Rechteerweiterung aktivieren.

- 1.1. Wechseln Sie in das Arbeitsverzeichnis `/home/student/playbook-multi`.

```
[student@workstation ~]$ cd ~/playbook-multi  
[student@workstation playbook-multi]$
```

- 1.2. Erstellen und öffnen Sie das neue Playbook `/home/student/playbook-multi/intranet.yml`, und fügen Sie am Anfang der Datei eine neue Zeile hinzu, die aus drei Bindestrichen besteht, um den Beginn einer YAML-Datei anzugeben.

- ```
---
```
- 1.3. Fügen Sie die folgende Zeile zur Datei `/home/student/playbook-multi/intranet.yml` hinzu, um den Beginn eines Plays mit dem Namen `Enable intranet services` zu signalisieren.

```
- name: Enable intranet services
```

14. Fügen Sie die folgende Zeile hinzu, um anzugeben, dass sich das Play auf den verwalteten Host `servera.lab.example.com` bezieht. Rücken Sie die Zeile mit zwei Leerzeichen ein (zur Ausrichtung am Keyword `name` über dieser Zeile), um anzugeben, dass sie Bestandteil des ersten Plays ist.

```
hosts: servera.lab.example.com
```

15. Fügen Sie die folgende Zeile hinzu, um die Rechteerweiterung zu aktivieren. Rücken Sie die Zeile mit zwei Leerzeichen ein (zur Ausrichtung am Keyword über dieser Zeile), um anzugeben, dass sie Bestandteil des ersten Plays ist.

```
become: yes
```

16. Fügen Sie die folgende Zeile hinzu, um den Beginn der Liste `tasks` zu definieren. Rücken Sie die Zeile mit zwei Leerzeichen ein (zur Ausrichtung am Keyword über dieser Zeile), um anzugeben, dass sie Bestandteil des ersten Plays ist.

```
tasks:
```

- 2. Definieren Sie als erste Aufgabe im ersten Play eine Aufgabe, mit der sichergestellt wird, dass sich die Pakete `httpd` und `firewalld` auf dem neuesten Stand befinden. Rücken Sie die erste Zeile der Aufgabe unbedingt mit vier Leerzeichen ein. Fügen Sie unter dem Keyword `tasks` im ersten Play die folgenden Zeilen hinzu.

```
- name: latest version of httpd and firewalld installed
  yum:
    name:
      - httpd
      - firewalld
    state: latest
```

Die erste Zeile enthält einen beschreibenden Namen für die Aufgabe. Die zweite Zeile ist mit sechs Leerzeichen eingerückt und ruft das Modul `yum` auf. Die nächste Zeile ist mit acht Leerzeichen eingerückt und enthält ein `name`-Keyword. Damit wird dem Modul `yum` mitgeteilt, bei welchen Paketen sichergestellt werden soll, dass sie auf dem neuesten Stand sind. Das Keyword `name` des Moduls `yum` (die sich vom Namen der Aufgabe unterscheidet) kann eine Liste von Paketen enthalten, die in den beiden folgenden Zeilen mit zehn Leerzeichen eingerückt ist. Nach der Liste teilt das mit acht Leerzeichen eingerückte Keyword `state` dem Modul `yum` mit, dass die neueste Version der Pakete installiert ist.

- 3. Fügen Sie eine Aufgabe zur Liste des ersten Plays hinzu, mit der sichergestellt wird, dass `/var/www/html/index.html` den richtigen Inhalt aufweist.

Fügen Sie die folgenden Zeilen hinzu, um den Inhalt für `/var/www/html/index.html` zu definieren. Rücken Sie die erste Zeile unbedingt mit vier Leerzeichen ein.

```
- name: test html page is installed
  copy:
    content: "Welcome to the example.com intranet!\n"
    dest: /var/www/html/index.html
```

Der erste Eintrag enthält einen beschreibenden Namen für die Aufgabe. Der zweite Eintrag ist mit sechs Leerzeichen eingerückt und ruft das Modul `copy` auf. Die verbleibenden Einträge sind mit acht Leerzeichen eingerückt und übergeben die erforderlichen Argumente, um sicherzustellen, dass die Webseite den richtigen Inhalt aufweist.

- 4. Definieren Sie in dem Play zwei weitere Aufgaben, um sicherzustellen, dass der Service `firewalld` ausgeführt und beim Booten gestartet wird und dass Verbindungen mit dem Service `httpd` zulässig sind.

- 4.1. Fügen Sie die folgenden Zeilen hinzu, um einen aktiven und laufenden `firewalld`-Service zu gewährleisten. Rücken Sie die erste Zeile unbedingt mit vier Leerzeichen ein.

```
- name: firewalld enabled and running
  service:
    name: firewalld
    enabled: true
    state: started
```

Der erste Eintrag enthält einen beschreibenden Namen für die Aufgabe. Der zweite Eintrag ist mit acht Leerzeichen eingerückt und ruft das Modul `service` auf. Die verbleibenden Einträge sind mit zehn Leerzeichen eingerückt und übergeben die erforderlichen Argumente, um sicherzustellen, dass der `firewalld`-Service aktiviert und gestartet wurde.

- 4.2. Fügen Sie die folgenden Zeilen hinzu, um sicherzustellen, dass `firewalld` HTTP-Verbindungen von Remote-Systemen zulässt. Rücken Sie die erste Zeile unbedingt mit vier Leerzeichen ein.

```
- name: firewalld permits access to httpd service
  firewalld:
    service: http
    permanent: true
    state: enabled
    immediate: yes
```

Der erste Eintrag enthält einen beschreibenden Namen für die Aufgabe. Der zweite Eintrag ist mit sechs Leerzeichen eingerückt und ruft das Modul `firewalld` auf. Die verbleibenden Einträge sind mit acht Leerzeichen eingerückt und übergeben die erforderlichen Argumente, um den Zugang zu Remote-HTTP-Verbindungen dauerhaft zu ermöglichen.

- 5. Fügen Sie eine abschließende Aufgabe zum ersten Play hinzu, mit der sichergestellt wird, dass der Service `httpd` ausgeführt und beim Booten gestartet wird.

Fügen Sie die folgenden Zeilen hinzu, um einen aktiven und laufenden `httpd`-Service zu gewährleisten. Rücken Sie die erste Zeile unbedingt mit vier Leerzeichen ein.

```
- name: httpd enabled and running
  service:
    name: httpd
    enabled: true
    state: started
```

Der erste Eintrag enthält einen beschreibenden Namen für die Aufgabe. Der zweite Eintrag ist mit sechs Leerzeichen eingerückt und ruft das Modul `service` auf. Die verbleibenden Einträge sind mit acht Leerzeichen eingerückt und übergeben die erforderlichen Argumente, um sicherzustellen, dass der `httpd`-Service aktiviert ist und ausgeführt wird.

- 6. Definieren Sie in `/home/student/playbook-multi/intranet.yml` ein zweites Play für `localhost`, mit dem der Intranet-Webserver getestet wird. Es benötigt keine Rechteerweiterung.

- 6.1. Fügen Sie die folgende Zeile hinzu, um den Beginn eines zweiten Plays zu definieren. Beachten Sie, dass es keine Einrückung gibt.

```
- name: Test intranet web server
```

- 6.2. Fügen Sie die folgende Zeile hinzu, um anzugeben, dass sich das Play auf den verwalteten Host `localhost` bezieht. Rücken Sie die Zeile unbedingt mit zwei Leerzeichen ein, um anzugeben, dass sie ein Teil des zweiten Plays ist.

```
hosts: localhost
```

- 6.3. Fügen Sie die folgende Zeile hinzu, um die Rechteerweiterung zu deaktivieren. Passen Sie die Einrückung am Keyword `hosts` darüber an.

```
become: no
```

- 6.4. Fügen Sie die folgende Zeile zur Datei `/home/student/playbook-multi/intranet.yml` hinzu, um den Beginn der Liste `tasks` zu definieren. Rücken Sie die Zeile unbedingt mit zwei Leerzeichen ein, um anzugeben, dass sie ein Teil des zweiten Plays ist.

```
tasks:
```

- 7. Fügen Sie dem zweiten Play eine einzelne Aufgabe hinzu und verwenden Sie das Modul `uri`, um Inhalt aus `http://servera.lab.example.com` anzufordern. Die Aufgabe sollte verifizieren, dass der HTTP-Statuscode 200 zurückgegeben wird. Konfigurieren Sie die Aufgabe so, dass der zurückgegebene Inhalt in die Aufgabenergebnis-Variable eingefügt wird.

Fügen Sie die folgenden Zeilen hinzu, um die Aufgabe zu erstellen, mit der der Webservice über den Kontrollknoten verifiziert werden. Rücken Sie die erste Zeile unbedingt mit vier Leerzeichen ein.

```
- name: connect to intranet web server
  uri:
    url: http://servera.lab.example.com
    return_content: yes
    status_code: 200
```

Die erste Zeile enthält einen beschreibenden Namen für die Aufgabe. Die zweite Zeile ist mit sechs Leerzeichen eingerückt und ruft das Modul `uri` auf. Die verbleibenden Zeilen sind mit sechs Leerzeichen eingerückt und übergeben die erforderlichen Argumente, um über den Kontrollknoten eine Abfrage zum Webinhalt an den verwalteten Host auszuführen und den erhaltenen Statuscode zu verifizieren. Das Keyword `return_content` stellt sicher, dass den Aufgabenergebnissen die Antwort des Servers hinzugefügt wird.

- 8. Verifizieren Sie, dass das finale Playbook `/home/student/playbook-multi/intranet.yml` den folgenden strukturierten Inhalt anzeigt. Speichern Sie dann, und schließen Sie die Datei.

```
---
- name: Enable intranet services
  hosts: servera.lab.example.com
  become: yes
  tasks:
    - name: latest version of httpd and firewalld installed
      yum:
        name:
          - httpd
          - firewalld
        state: latest

    - name: test html page is installed
      copy:
        content: "Welcome to the example.com intranet!\n"
        dest: /var/www/html/index.html

    - name: firewalld enabled and running
      service:
        name: firewalld
        enabled: true
        state: started

    - name: firewalld permits access to httpd service
      firewalld:
        service: http
        permanent: true
        state: enabled
        immediate: yes

    - name: httpd enabled and running
      service:
        name: httpd
        enabled: true
        state: started
```

**Kapitel 2 |** Implementieren eines Ansible-Playbooks

```
- name: Test intranet web server
hosts: localhost
become: no
tasks:
- name: connect to intranet web server
uri:
url: http://servera.lab.example.com
return_content: yes
status_code: 200
```

- 9. Führen Sie den Befehl `ansible-playbook --syntax-check` aus, um die Syntax des Playbooks `/home/student/playbook-multi/intranet.yml` zu überprüfen.

```
[student@workstation playbook-multi]$ ansible-playbook --syntax-check intranet.yml
playbook: intranet.yml
```

- 10. Führen Sie das Playbook mit der Option `-v` aus, um detaillierte Ergebnisse für jede Aufgabe auszugeben. Überprüfen Sie die generierte Ausgabe, um den erfolgreichen Abschluss aller Aufgaben zu gewährleisten. Verifizieren Sie, dass eine HTTP GET-Anforderung an `http://servera.lab.example.com` den richtigen Inhalt bereitstellt.

```
[student@workstation playbook-multi]$ ansible-playbook -v intranet.yml
...output omitted...

PLAY [Enable intranet services] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [latest version of httpd and firewalld installed] ****
changed: [servera.lab.example.com] => {"changed": true, ...output omitted...

TASK [test html page is installed] ****
changed: [servera.lab.example.com] => {"changed": true, ...output omitted...

TASK [firewalld enabled and running] ****
ok: [servera.lab.example.com] => {"changed": false, ...output omitted...

TASK [firewalld permits http service] ****
changed: [servera.lab.example.com] => {"changed": true, ...output omitted...

TASK [httpd enabled and running] ****
changed: [servera.lab.example.com] => {"changed": true, ...output omitted...

PLAY [Test intranet web server] ****
TASK [Gathering Facts] ****
ok: [localhost]

TASK [connect to intranet web server] ****
ok: [localhost] => {"accept_ranges": "bytes", "changed": false, "connection": "close", "content": "Welcome to the example.com intranet!\n", "content_length": 1}
```

```
"37", "content_type": "text/html; charset=UTF-8", "cookies": {}, "cookies_string": "", "date": "...output omitted...", "etag": "\"25-5790ddbcc5a48\"", "last_modified": "...output omitted...", "msg": "OK (37 bytes)", "redirected": false, "server": "Apache/2.4.6 (Red Hat Enterprise Linux)", "status": 200, "url": "http://servera.lab.example.com"}②  
  
PLAY RECAP ****  
localhost : ok=2    changed=0    unreachable=0    failed=0  
servera.lab.example.com : ok=6    changed=4    unreachable=0    failed=0
```

- ① Der Server hat mit dem gewünschten Inhalt `Welcome to the example.com intranet!`\n geantwortet.
- ② Der Server hat mit dem HTTP-Statuscode 200 geantwortet.

## Beenden

Führen Sie auf `workstation` den Befehl `lab playbook-multi finish` aus, um die in dieser Übung erstellten Ressourcen zu bereinigen.

```
[student@workstation ~]$ lab playbook-multi finish
```

Hiermit ist die angeleitete Übung beendet.

## ► Praktische Übung

# Implementieren von Playbooks

### Leistungscheckliste

In diesem Lab konfigurieren und führen Sie administrative Aufgaben auf verwalteten Hosts mit einem Playbook aus.

### Ergebnisse

Sie sollten in der Lage sein, ein Playbook zu erstellen und auszuführen, um Web- und Datenbankservices auf einem verwalteten Host zu installieren, zu konfigurieren und ihren Status zu verifizieren.

### Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab playbook-review start` aus. Diese Funktion stellt sicher, dass der verwaltete Host `serverb.lab.example.com` im Netzwerk erreichbar ist. Dies gewährleistet auch die Installation der korrekten Ansible-Konfigurationsdatei und Inventardatei auf dem Kontrollknoten.

```
[student@workstation ~]$ lab playbook-review start
```

Auf `workstation` wurde das Arbeitsverzeichnis `/home/student/playbook-review` für das Ansible-Projekt erstellt. Das Verzeichnis wurde bereits mit der Konfigurationsdatei `ansible.cfg` und der Datei `inventory` gefüllt. In der Inventardatei ist der verwaltete Host `serverb.lab.example.com` bereits definiert.

### Anweisungen



#### Anmerkung

Das Playbook, das in dieser praktischen Übung verwendet wird, ist dem Playbook sehr ähnlich, das in der vorherigen angeleiteten Übung in diesem Kapitel erstellt wurde. Wenn Sie das Playbook dieser praktischen Übung nicht von Neuem erstellen möchten, können Sie das Playbook der vorherigen Übung als Ausgangspunkt für diese praktische Übung verwenden.

Achten Sie in diesem Fall darauf, die richtigen Hosts als Ziel zu wählen und die Aufgaben so zu ändern, dass sie den Anweisungen für diese Übung entsprechen.

1. Erstellen Sie das neue Playbook `/home/student/playbook-review/internet.yml`, und fügen Sie die erforderlichen Einträge hinzu, um ein erstes Play mit dem Namen `Enable internet services` zu erstellen und seinen beabsichtigten verwalteten Host `serverb.lab.example.com` festzulegen. Fügen Sie einen Eintrag zur Aktivierung der Rechteerweiterung und einen zum Start einer Aufgabenliste hinzu.

2. Fügen Sie die erforderlichen Einträge zur Datei `/home/student/playbook-review/internet.yml` hinzu, um eine Aufgabe zu definieren, die die neuesten Versionen von `firewalld-`, `httpd-`, `mariadb-server-`, `php-` und `php-mysqlnd`-Paketen installiert.
3. Fügen Sie die erforderlichen Einträge zur Datei `/home/student/playbook-review/internet.yml` hinzu, um die Aufgaben zur Konfiguration der Firewall festzulegen. Sie sollten dafür sorgen, dass der `firewalld`-Service aktiviert ist und ausgeführt wird und der `http`-Service Zugriff hat.
4. Fügen Sie die erforderlichen Aufgaben hinzu, um sicherzustellen, dass die Services `httpd` und `mariadb` aktiviert sind und ausgeführt werden.
5. Fügen Sie die erforderlichen Einträge hinzu, um die finale Aufgabe zur Generierung von Webinhalten zu Testzwecken zu definieren. Verwenden Sie das Modul `get_url`, um `http://materials.example.com/labs/playbook-review/index.php` in `/var/www/html/` auf dem verwalteten Host zu kopieren.
6. Bestimmen Sie in `/home/student/playbook-review/internet.yml` ein weiteres Play für die Aufgabe, die auf dem Kontrollknoten ausgeführt werden soll. Dieses Play testet den Zugriff auf den Webserver, der auf dem verwalteten Host `serverb` ausgeführt werden soll. Dieses Play erfordert keine Rechteerweiterung und wird auf dem verwalteten Host `localhost` ausgeführt.
7. Fügen Sie eine Aufgabe hinzu, mit der der Webservice getestet wird, der auf `serverb` vom Kontrollknoten mit dem Modul `uri` ausgeführt wird. Überprüfen Sie den Rückgabestatuscode von `200`.
8. Verifizieren Sie die Syntax des Playbooks `internet.yml`.
9. Führen Sie mit dem Befehl `ansible-playbook` das Playbook aus. Überprüfen Sie die generierte Ausgabe, um den erfolgreichen Abschluss aller Aufgaben zu gewährleisten.

## Bewertung

Bewerten Sie Ihre Arbeit, indem Sie auf Ihrem `workstation`-Rechner den Befehl `lab playbook-review grade` ausführen. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie das Skript so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab playbook-review grade
```

## Beenden

Führen Sie auf `workstation` das Skript `lab playbook-review finish` aus, um die in dieser praktischen Übung erstellten Ressourcen zu bereinigen.

```
[student@workstation ~]$ lab playbook-review finish
```

Hiermit ist die praktische Übung abgeschlossen.

## ► Lösung

# Implementieren von Playbooks

### Leistungscheckliste

In diesem Lab konfigurieren und führen Sie administrative Aufgaben auf verwalteten Hosts mit einem Playbook aus.

### Ergebnisse

Sie sollten in der Lage sein, ein Playbook zu erstellen und auszuführen, um Web- und Datenbankservices auf einem verwalteten Host zu installieren, zu konfigurieren und ihren Status zu verifizieren.

### Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab playbook-review start` aus. Diese Funktion stellt sicher, dass der verwaltete Host `serverb.lab.example.com` im Netzwerk erreichbar ist. Dies gewährleistet auch die Installation der korrekten Ansible-Konfigurationsdatei und Inventardatei auf dem Kontrollknoten.

```
[student@workstation ~]$ lab playbook-review start
```

Auf `workstation` wurde das Arbeitsverzeichnis `/home/student/playbook-review` für das Ansible-Projekt erstellt. Das Verzeichnis wurde bereits mit der Konfigurationsdatei `ansible.cfg` und der Datei `inventory` gefüllt. In der Inventardatei ist der verwaltete Host `serverb.lab.example.com` bereits definiert.

### Anweisungen



#### Anmerkung

Das Playbook, das in dieser praktischen Übung verwendet wird, ist dem Playbook sehr ähnlich, das in der vorherigen angeleiteten Übung in diesem Kapitel erstellt wurde. Wenn Sie das Playbook dieser praktischen Übung nicht von Neuem erstellen möchten, können Sie das Playbook der vorherigen Übung als Ausgangspunkt für diese praktische Übung verwenden.

Achten Sie in diesem Fall darauf, die richtigen Hosts als Ziel zu wählen und die Aufgaben so zu ändern, dass sie den Anweisungen für diese Übung entsprechen.

1. Erstellen Sie das neue Playbook `/home/student/playbook-review/internet.yml`, und fügen Sie die erforderlichen Einträge hinzu, um ein erstes Play mit dem Namen `Enable internet services` zu erstellen und seinen beabsichtigten verwalteten Host `serverb.lab.example.com` festzulegen. Fügen Sie einen Eintrag zur Aktivierung der Rechteerweiterung und einen zum Start einer Aufgabenliste hinzu.

**Kapitel 2 |** Implementieren eines Ansible-Playbooks

- 1.1. Fügen Sie am Anfang von `/home/student/playbook-review/internet.yml` den folgenden Eintrag hinzu, um mit dem YAML-Format zu beginnen.

```
--
```

- 1.2. Fügen Sie den folgenden Eintrag hinzu, um den Beginn eines Plays mit dem Namen `Enable internet services` zu signalisieren.

```
- name: Enable internet services
```

- 1.3. Fügen Sie den folgenden Eintrag hinzu, um anzugeben, dass sich das Play auf den verwalteten Host `serverb` bezieht.

```
hosts: serverb.lab.example.com
```

- 1.4. Fügen Sie den folgenden Eintrag hinzu, um die Rechteerweiterung zu aktivieren.

```
become: yes
```

- 1.5. Fügen Sie den folgenden Eintrag hinzu, um den Beginn der Liste `tasks` zu definieren.

```
tasks:
```

2. Fügen Sie die erforderlichen Einträge zur Datei `/home/student/playbook-review/internet.yml` hinzu, um eine Aufgabe zu definieren, die die neuesten Versionen von `firewalld`-, `httpd`-, `mariadb-server`-, `php`- und `php-mysqld`-Paketen installiert.

```
- name: latest version of all required packages installed
  yum:
    name:
      - firewalld
      - httpd
      - mariadb-server
      - php
      - php-mysqld
    state: latest
```

3. Fügen Sie die erforderlichen Einträge zur Datei `/home/student/playbook-review/internet.yml` hinzu, um die Aufgaben zur Konfiguration der Firewall festzulegen. Sie sollten dafür sorgen, dass der `firewalld`-Service aktiviert ist und ausgeführt wird und der `http`-Service Zugriff hat.

```
- name: firewalld enabled and running
  service:
    name: firewalld
    enabled: true
    state: started

- name: firewalld permits http service
  firewalld:
    service: http
```

```
permanent: true
state: enabled
immediate: yes
```

4. Fügen Sie die erforderlichen Aufgaben hinzu, um sicherzustellen, dass die Services `httpd` und `mariadb` aktiviert sind und ausgeführt werden.

```
- name: httpd enabled and running
  service:
    name: httpd
    enabled: true
    state: started

- name: mariadb enabled and running
  service:
    name: mariadb
    enabled: true
    state: started
```

5. Fügen Sie die erforderlichen Einträge hinzu, um die finale Aufgabe zur Generierung von Webinhalten zu Testzwecken zu definieren. Verwenden Sie das Modul `get_url`, um `http://materials.example.com/labs/playbook-review/index.php` in `/var/www/html/` auf dem verwalteten Host zu kopieren.

```
- name: test php page is installed
  get_url:
    url: "http://materials.example.com/labs/playbook-review/index.php"
    dest: /var/www/html/index.php
    mode: 0644
```

6. Bestimmen Sie in `/home/student/playbook-review/internet.yml` ein weiteres Play für die Aufgabe, die auf dem Kontrollknoten ausgeführt werden soll. Dieses Play testet den Zugriff auf den Webserver, der auf dem verwalteten Host `serverb` ausgeführt werden soll. Dieses Play erfordert keine Rechteerweiterung und wird auf dem verwalteten Host `localhost` ausgeführt.

- 6.1. Fügen Sie den folgenden Eintrag hinzu, um den Beginn eines zweiten Plays mit dem Namen `Test internet web server` zu signalisieren.

```
- name: Test internet web server
```

- 6.2. Fügen Sie den folgenden Eintrag hinzu, um anzugeben, dass sich das Play auf den verwalteten Host `localhost` bezieht.

```
hosts: localhost
```

- 6.3. Fügen Sie nach dem Keyword `hosts` die folgende Zeile hinzu, um die Rechteerweiterung für das zweite Play zu deaktivieren.

```
become: no
```

- 6.4. Fügen Sie der Datei `/home/student/playbook-review/internet.yml` einen Eintrag hinzu, um den Beginn der Liste `tasks` festzulegen.

```
tasks:
```

7. Fügen Sie eine Aufgabe hinzu, mit der der Webservice getestet wird, der auf `serverb` vom Kontrollknoten mit dem Modul `uri` ausgeführt wird. Überprüfen Sie den Rückgabestatuscode von 200.

```
- name: connect to internet web server
  uri:
    url: http://serverb.lab.example.com
    status_code: 200
```

8. Verifizieren Sie die Syntax des Playbooks `internet.yml`.

```
[student@workstation playbook-review]$ ansible-playbook --syntax-check \
> internet.yml

playbook: internet.yml
```

9. Führen Sie mit dem Befehl `ansible-playbook` das Playbook aus. Überprüfen Sie die generierte Ausgabe, um den erfolgreichen Abschluss aller Aufgaben zu gewährleisten.

```
[student@workstation playbook-review]$ ansible-playbook internet.yml
PLAY [Enable internet services] ****

TASK [Gathering Facts] ****
ok: [serverb.lab.example.com]

TASK [latest version of all required packages installed] ****
changed: [serverb.lab.example.com]

TASK [firewalld enabled and running] ****
ok: [serverb.lab.example.com]

TASK [firewalld permits http service] ****
changed: [serverb.lab.example.com]

TASK [httpd enabled and running] ****
changed: [serverb.lab.example.com]

TASK [mariadb enabled and running] ****
changed: [serverb.lab.example.com]

TASK [test php page installed] ****
changed: [serverb.lab.example.com]

PLAY [Test internet web server] ****

TASK [Gathering Facts] ****
ok: [localhost]

TASK [connect to internet web server] ****
ok: [localhost]
```

```
PLAY RECAP ****
localhost           : ok=2      changed=0      unreachable=0      failed=0
serverb.lab.example.com  : ok=7      changed=5      unreachable=0      failed=0
```

## Bewertung

Bewerten Sie Ihre Arbeit, indem Sie auf Ihrem `workstation`-Rechner den Befehl `lab playbook-review grade` ausführen. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie das Skript so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab playbook-review grade
```

## Beenden

Führen Sie auf `workstation` das Skript `lab playbook-review finish` aus, um die in dieser praktischen Übung erstellten Ressourcen zu bereinigen.

```
[student@workstation ~]$ lab playbook-review finish
```

Hiermit ist die praktische Übung abgeschlossen.

# Zusammenfassung

---

In diesem Kapitel wurden die folgenden Themen behandelt:

- Ein *Play* stellt eine geordnete Liste von Aufgaben dar, die auf aus dem Inventar ausgewählten Hosts ausgeführt werden.
- Ein *Playbook* ist eine Textdatei, die eine Liste mit einem oder mehreren Plays enthält, die in der entsprechenden Reihenfolge ausgeführt werden sollen.
- Ansible-Playbooks sind im YAML-Format geschrieben.
- YAML-Dateien sind mit Leerzeicheneinzügen strukturiert, um die Datenhierarchie anzuzeigen.
- Aufgaben werden mit standardisiertem Code implementiert, der in Ansible-Modulen zusammengefasst wird.
- Mit dem Befehl `ansible-doc` können installierte Module aufgelistet und die Dokumentation sowie Beispielcodeauszüge zu ihrer Nutzung in Playbooks bereitgestellt werden.
- Der Befehl `ansible-playbook` wird zur Verifizierung der Playbook-Syntax und zur Ausführung von Playbooks verwendet.



## Kapitel 3

# Verwalten von Variablen und Fakten

### Ziel

Playbooks erstellen, die Variablen verwenden, um die Verwaltung des Playbooks und der Fakten zu vereinfachen, um Informationen zu den verwalteten Hosts zu referenzieren.

### Ziele

- Erstellen und Verweisen auf Variablen, die bestimmte Hosts oder Hostgruppen, das Play oder die globale Umgebung betreffen, und Beschreibung der Priorisierung von Variablen.
- Vertrauliche Variablen mit Ansible Vault verschlüsseln und Playbooks ausführen, die auf Vault-kodierte Dateien verweisen.
- Daten zu verwalteten Hosts mithilfe von Ansible-Fakten referenzieren und benutzerdefinierte Fakten zu verwalteten Hosts konfigurieren.

### Abschnitte

- Verwalten von Variablen (und angeleitete Übung)
- Verwalten von Secrets (und angeleitete Übung)
- Verwalten von Fakten (und angeleitete Übung)

### Praktische Übung

- Verwalten von Variablen und Fakten

# Verwalten von Variablen

---

## Ziele

Am Ende dieses Abschnitts sollten Sie in der Lage sein, Variablen zu erstellen und zu referenzieren, die sich auf bestimmte Hosts oder Hostgruppen, das Play oder die globale Umgebung auswirken, sowie die Funktionsweise der Variablenpriorität zu beschreiben.

## Einführung in Ansible-Variablen

Ansible unterstützt Variablen, in denen Werte gespeichert werden können, um sie dann in Dateien im Ansible-Projekt wiederzuverwenden. Dies kann die Erstellung und Wartung eines Projekts vereinfachen und die Anzahl an Fehlern reduzieren.

Variablen bieten eine praktische Möglichkeit, dynamische Werte für eine bestimmte Umgebung in Ihrem Ansible-Projekt zu verwalten. Beispiele für Werte, die in Variablen enthalten sein können:

- Benutzer, die erstellt werden sollen
- Pakete, die installiert werden sollen
- Services, die neu gestartet werden sollen
- Services, die entfernt werden sollen
- Archive, die aus dem Internet abgerufen werden sollen

## Benennen von Variablen

Variablennamen müssen mit einem Buchstaben beginnen. Zudem dürfen sie nur Buchstaben, Zahlen und Unterstriche enthalten.

In der folgenden Tabelle wird der Unterschied zwischen ungültigen und gültigen Variablennamen veranschaulicht.

### Beispiele für ungültige und gültige Ansible-Variablennamen

| Ungültige Variablennamen | Gültige Variablennamen            |
|--------------------------|-----------------------------------|
| web server               | web_server                        |
| remote.file              | remote_file                       |
| 1st file                 | file_1<br>file1                   |
| remoteserver\$1          | remote_server_1<br>remote_server1 |

## Definieren von Variablen

Variablen können an vielen Stellen in einem Ansible-Projekt definiert werden. Wenn eine Variable mit dem gleichen Namen an zwei Stellen gesetzt wird und diese Einstellungen unterschiedliche Werte aufweisen, bestimmt die *Rangordnung*, welcher Wert verwendet wird.

Sie können eine Variable festlegen, die sich auf eine Gruppe von Hosts oder nur auf einzelne Hosts auswirkt. Einige Variablen sind *Fakten*, die von Ansible basierend auf der Konfiguration eines Systems festgelegt werden können. Andere Variablen können innerhalb des Playbooks festgelegt werden und wirken sich auf ein Play in diesem Playbook oder nur auf eine Aufgabe in diesem Playbook aus. Sie können auch *zusätzliche Variablen* in der Befehlszeile von `ansible-playbook` setzen, indem Sie die Option `--extra-vars` oder `-e` verwenden und diese Variablen angeben. Sie überschreiben alle anderen Werte für diesen Variablennamen.

Hier finden Sie eine vereinfachte Liste von Möglichkeiten, um eine Variable zu definieren (geordnet von der niedrigsten zur höchsten Priorität):

- Im Inventar definierte Gruppenvariablen
- Gruppenvariablen, die in Dateien in einem Unterverzeichnis `group_vars` im gleichen Verzeichnis wie das Inventar oder das Playbook definiert sind
- Im Inventar definierte Hostvariablen
- Hostvariablen, die in Dateien in einem Unterverzeichnis `host_vars` im gleichen Verzeichnis wie das Inventar oder das Playbook definiert sind
- Hostfakten, die zur Laufzeit erkannt werden
- Play-Variablen im Playbook (`vars` und `vars_files`)
- Aufgabenvariablen
- Zusätzliche Variablen, die in der Befehlszeile definiert sind

Beispielsweise wird eine Variable, die so festgelegt ist, dass sie sich auf die gesamte Hostgruppe auswirkt, durch eine Variable überschrieben, die den gleichen Namen aufweist und so festgelegt ist, dass sie sich auf einen einzelnen Host auswirkt.

Eine empfohlene Vorgehensweise besteht darin, global eindeutige Variablennamen zu wählen, damit Sie keine Rangordnungsregeln berücksichtigen müssen. Manchmal möchten Sie jedoch vielleicht die Priorität verwenden, um verschiedene Hosts oder Hostgruppen zu veranlassen, andere Einstellungen als Ihre Standardeinstellungen abzurufen.

Wenn derselbe Variablenname auf mehreren Ebenen definiert ist, hat die höchste Ebene Priorität. Ein enger Bereich, wie eine Hostvariable oder eine Aufgabenvariable, hat Vorrang vor einem umfangreicheren Bereich, wie einer Gruppenvariablen oder einer Play-Variablen. Durch das Inventar definierte Variablen werden durch Variablen überschrieben, die durch das Playbook definiert werden. Zusätzliche Variablen, die in der Befehlszeile mit der Option `--extra-vars` oder `-e` definiert sind, haben die höchste Priorität.

Eine ausführliche und genauere Erörterung der Variablenpriorität finden Sie in der Ansible-Dokumentation unter Variablenpriorität: Wo sollte ich eine Variable platzieren? [[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_variables.html#variable-precedence-where-should-i-put-a-variable](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable)].

## Variablen in Playbooks

Variablen spielen in Ansible Playbooks eine wichtige Rolle, da sie die Verwaltung variabler Daten in einem Playbook erleichtern.

### Definieren von Variablen in Playbooks

Beim Schreiben von Playbooks können Sie Ihre eigenen Variablen definieren und diese Werte dann in einer Aufgabe aufrufen. Beispielsweise kann eine Variable mit dem Namen `web_package` mit dem Wert `httpd` definiert werden. Eine Aufgabe kann die Variable anschließend mit dem Modul `yum` aufrufen, um das Paket `httpd` zu installieren.

Playbook-Variablen können auf vielfältige Weise definiert werden. Eine gängige Methode besteht darin, eine Variable im Block `vars` am Anfang eines Playbooks zu platzieren:

```
- hosts: all
  vars:
    user: joe
    home: /home/joe
```

Playbook-Variablen können auch in externen Dateien definiert werden. In diesem Fall kann anstelle des Blocks `vars` im Playbook die Anweisung `vars_files` verwendet werden, gefolgt von einer Liste der Namen für externe Variablendateien, mit Bezug zum Speicherort des Playbooks.

```
- hosts: all
  vars_files:
    - vars/users.yml
```

Die Playbook-Variablen werden dann in dieser Datei oder diesen Dateien im YAML-Format definiert:

```
user: joe
home: /home/joe
```

### Verwenden von Variablen in Playbooks

Sobald Variablen deklariert wurden, können Administratoren diese Variablen in Aufgaben verwenden. Variablen werden referenziert, indem der Variablenname in doppelte geschweifte Klammern (`{} {}`) eingeschlossen wird. Ansible ersetzt bei der Ausführung der Aufgaben die Variable durch deren Wert.

```
vars:
  user: joe

tasks:
  # This line will read: Creates the user joe
  - name: Creates the user {{ user }}
    user:
      # This line will create the user named Joe
      name: "{{ user }}"
```



### Wichtig

Wenn eine Variable als erstes Element zum Festlegen eines Wertes verwendet wird, muss sie in Anführungszeichen gesetzt werden. Dadurch wird verhindert, dass Ansible die Variablenreferenz als Start eines YAML-Dictionary interpretiert. Wenn Anführungszeichen fehlen, wird die folgende Meldung angezeigt:

```
yum:  
  name: {{ service }}  
    ^ here  
We could be wrong, but this one looks like it might be an issue with  
missing quotes. Always quote template expression brackets when they  
start a value. For instance:  
  
with_items:  
  - {{ foo }}  
  
Should be written as:  
  
with_items:  
  - "{{ foo }}"
```

## Hostvariablen und Gruppenvariablen

Inventarvariablen, die direkt auf Hosts angewendet werden, lassen sich in zwei grobe Kategorien unterteilen: *Hostvariablen*, die auf einen bestimmten Host angewendet werden, und *Gruppenvariablen*, die auf alle Hosts in einer Hostgruppe oder in einer Gruppe von Hostgruppen angewendet werden. Hostvariablen haben Priorität vor Gruppenvariablen, aber Variablen, die in einem Playbook definiert sind, haben Priorität vor beiden.

Eine Möglichkeit, Hostvariablen und Gruppenvariablen zu definieren, besteht darin, sie direkt in der Inventardatei anzugeben. Dies ist ein älterer Ansatz und nicht der einfachste, aber Sie könnten trotzdem auf ihn stoßen, weil dabei alle Inventarinformationen und Variableneinstellungen für Hosts und Hostgruppen in einer Datei zusammengefasst werden.

- Definieren der Hostvariablen `ansible_user` für `demo.example.com`:

```
[servers]  
demo.example.com ansible_user=joe
```

- Definieren der Gruppenvariablen `user` für die Hostgruppe `servers`.

```
[servers]  
demo1.example.com  
demo2.example.com  
  
[servers:vars]  
user=joe
```

- Definieren der Gruppenvariablen `user` für die Gruppe `servers`, die aus zwei Hostgruppen besteht, die jeweils über zwei Server verfügen.

```
[servers1]
demo1.example.com
demo2.example.com

[servers2]
demo3.example.com
demo4.example.com

[servers:children]
servers1
servers2

[servers:vars]
user=joe
```

Dieser Ansatz hat unter anderem die folgenden Nachteile: Die Arbeit mit der Inventardatei wird erschwert, Informationen über Hosts und Variablen werden in derselben Datei verwendet und die Syntax ist veraltet.

## Verwenden von Verzeichnissen zum Auffüllen von Host- und Gruppenvariablen

Die bevorzugte Methode, Variablen für Hosts und Hostgruppen zu definieren, besteht darin, die zwei Verzeichnisse `group_vars` und `host_vars` im selben Arbeitsverzeichnis wie die Inventardatei oder das Playbook zu erstellen. Diese Verzeichnisse enthalten Dateien, die Gruppenvariablen bzw. Hostvariablen definieren.



### Wichtig

Das empfohlene Vorgehen ist, Inventarvariablen in den Verzeichnissen `host_vars` und `group_vars` zu definieren und sie nicht direkt in den Inventardateien festzulegen.

Um Gruppenvariablen für die Gruppe `servers` zu definieren, erstellen Sie eine YAML-Datei mit dem Namen `group_vars/servers`. Dann werden Variablen durch den Inhalt dieser Datei auf Werte festgelegt. Dabei wird die gleiche Syntax wie in einem Playbook verwendet:

```
user: joe
```

Erstellen Sie entsprechend, um Hostvariablen für einen bestimmten Host zu definieren, eine Datei mit einem Namen, der dem des Hosts im Verzeichnis `host_vars` entspricht, in dem die Hostvariablen enthalten sein sollen.

Die folgenden Beispiele demonstrieren dieses Vorgehen ausführlicher. Stellen Sie sich ein Szenario vor, in dem zwei Rechenzentren verwaltet werden müssen und die Rechenzentrum-Hosts in der Inventardatei `~/project/inventory` definiert sind:

```
[admin@station project]$ cat ~/project/inventory
[datacenter1]
demo1.example.com
demo2.example.com
```

**Kapitel 3 |** Verwalten von Variablen und Fakten

```
[datacenter2]
demo3.example.com
demo4.example.com

[datacenters:children]
datacenter1
datacenter2
```

- Wenn Sie einen allgemeinen Wert für alle Server in beiden Rechenzentren definieren müssen, legen Sie eine Gruppenvariable für die Hostgruppe `datacenters` fest:

```
[admin@station project]$ cat ~/project/group_vars/datacenters
package: httpd
```

- Wenn der festzulegende Wert sich je nach Rechenzentrum unterscheidet, legen Sie eine Gruppenvariable für die jeweilige Rechenzentrums-Hostgruppe fest:

```
[admin@station project]$ cat ~/project/group_vars/datacenter1
package: httpd
[admin@station project]$ cat ~/project/group_vars/datacenter2
package: apache
```

- Wenn der festzulegende Wert sich je nach Host in den jeweiligen Rechenzentren unterscheidet, sollten Sie die Variablen in separaten Hostvariablen dateien festlegen:

```
[admin@station project]$ cat ~/project/host_vars/demo1.example.com
package: httpd
[admin@station project]$ cat ~/project/host_vars/demo2.example.com
package: apache
[admin@station project]$ cat ~/project/host_vars/demo3.example.com
package: mariadb-server
[admin@station project]$ cat ~/project/host_vars/demo4.example.com
package: mysql-server
```

Die Verzeichnisstruktur für das Beispielprojekt `project`, sofern es alle obigen Beispieldateien enthält, würde wie folgt aussehen:

```
project
├── ansible.cfg
├── group_vars
│   ├── datacenters
│   │   ├── datacenters1
│   │   └── datacenters2
│   └── host_vars
│       ├── demo1.example.com
│       ├── demo2.example.com
│       ├── demo3.example.com
│       └── demo4.example.com
└── inventory
    └── playbook.yml
```



### Anmerkung

Ansible sucht nach den Unterverzeichnissen `host_vars` und `group_vars`, die relativ zum Inventar und zum Playbook sind. Wenn sich Ihr Inventar und Ihr Playbook zufällig im gleichen Verzeichnis befinden, sucht Ansible einfach in diesem Verzeichnis nach diesen Unterverzeichnissen. Wenn sich Ihr Inventar und Ihr Playbook in getrennten Verzeichnissen befinden, sucht Ansible an beiden Stellen nach den Unterverzeichnissen `host_vars` und `group_vars`. Die Playbook-Unterverzeichnisse haben höhere Priorität.

## Überschreiben von Variablen in der Befehlszeile

Inventarvariablen werden von in einem Playbook festgelegten Variablen überschrieben, aber beide Variablenarten können durch Argumente überschrieben werden, die in der Befehlszeile an die Befehle `ansible` oder `ansible-playbook` übergeben werden. Über die Befehlszeile festgelegte Variablen werden als *weitere Variablen* bezeichnet.

Weitere Variablen können nützlich sein, wenn Sie den definierten Wert für eine Variable für eine einmalige Ausführung eines Playbooks überschreiben müssen. Beispiel:

```
[user@demo ~]$ ansible-playbook main.yml -e "package=apache"
```

## Verwenden von Arrays als Variablen

Anstatt Konfigurationsdaten, die sich auf dasselbe Element (eine Liste von Paketen, eine Liste von Services, eine Liste von Benutzern usw.) beziehen, mehreren Variablen zuzuweisen, können Administratoren Arrays verwenden. Arrays bieten den Vorteil, dass sie durchsucht werden können.

Sehen Sie sich beispielsweise diesen Codeausschnitt an:

```
user1_first_name: Bob
user1_last_name: Jones
user1_home_dir: /users/bjones
user2_first_name: Anne
user2_last_name: Cook
user2_home_dir: /users/acock
```

Dieses Fragment könnte als Array mit dem Namen `users` umgeschrieben werden:

```
users:
bjones:
  first_name: Bob
  last_name: Jones
  home_dir: /users/bjones
acock:
  first_name: Anne
  last_name: Cook
  home_dir: /users/acock
```

Sie können die folgenden Variablen verwenden, um auf die Benutzerdaten zuzugreifen:

```
# Returns 'Bob'  
users.bjones.first_name  
  
# Returns '/users/acook'  
users.acook.home_dir
```

Da die Variable als ein Python-Verzeichnis definiert ist, steht eine alternative Syntax zur Verfügung.

```
# Returns 'Bob'  
users['bjones']['first_name']  
  
# Returns '/users/acook'  
users['acook']['home_dir']
```



### Wichtig

Die Punktnotation kann Probleme verursachen, wenn die Schlüsselnamen mit Namen von Python-Methoden oder -Attributnen, wie `discard`, `copy`, `add` usw., übereinstimmen. Durch die Verwendung der Notation mit eckigen Klammern können Konflikte und Fehler vermieden werden.

Beide Syntaxen sind gültig, aber um die Fehlerbehebung zu erleichtern, wird von Red Hat empfohlen, eine Syntax konsistent in allen Dateien eines Ansible-Projekts zu verwenden.

## Erfassen der Befehlausgabe mit registrierten Variablen

Sie können die Anweisung `register` verwenden, um die Ausgabe eines Befehls zu erfassen. Die Ausgabe wird in einer temporären Variable gespeichert, die später im Playbook zum Debuggen oder zu einem anderen Zweck, wie etwa einer bestimmte Konfiguration auf Basis der Ausgabe eines Befehls festzulegen, verwendet werden kann.

Das folgende Playbook demonstriert, wie die Ausgabe eines Befehls für das Debuggen erfasst werden kann:

```
---  
- name: Installs a package and prints the result  
hosts: all  
tasks:  
  - name: Install the package  
    yum:  
      name: httpd  
      state: installed  
    register: install_result  
  
  - debug:  
    var: install_result
```

Bei der Ausführung des Playbooks wird mit dem Modul `debug` der Wert der registrierten Variable `install_result` auf dem Terminal ausgegeben.

```
[user@demo ~]$ ansible-playbook playbook.yml
PLAY [Installs a package and prints the result] ****
TASK [setup] ****
ok: [demo.example.com]

TASK [Install the package] ****
ok: [demo.example.com]

TASK [debug] ****
ok: [demo.example.com] => {
    "install_result": {
        "changed": false,
        "msg": "",
        "rc": 0,
        "results": [
            "httpd-2.4.6-40.el7.x86_64 providing httpd is already installed"
        ]
    }
}
PLAY RECAP ****
demo.example.com      : ok=3      changed=0      unreachable=0      failed=0
```



## Literaturhinweise

### Inventory – Ansible-Dokumentation

[https://docs.ansible.com/ansible/2.9/user\\_guide/intro\\_inventory.html](https://docs.ansible.com/ansible/2.9/user_guide/intro_inventory.html)

### Variables – Ansible-Dokumentation

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_variables.html](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_variables.html)

### Variablenpriorität: Wo sollte ich eine Variable platzieren?

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_variables.html#variable-precedence-where-should-i-put-a-variable](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable)

### YAML-Syntax – Ansible-Dokumentation

[https://docs.ansible.com/ansible/2.9/reference\\_appendices/YAMLSyntax.html](https://docs.ansible.com/ansible/2.9/reference_appendices/YAMLSyntax.html)

## ► Angeleitete Übung

# Verwalten von Variablen

In dieser Übung definieren und verwenden Sie Variablen in einem Playbook.

## Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Definieren von Variablen in einem Playbook.
- Erstellen von Aufgaben, die definierte Variablen verwenden.

## Bevor Sie Beginnen

Melden Sie sich als student mit dem Passwort student bei workstation an.

Führen Sie auf workstation den Befehl `lab data-variables start` aus. Diese Funktion erstellt das Arbeitsverzeichnis `data-variables` und füllt es mit einer Ansible-Konfigurationsdatei und dem Hostinventar.

```
[student@workstation ~]$ lab data-variables start
```

## Anweisungen

- 1. Wechseln Sie auf workstation als Benutzer student ins Verzeichnis `/home/student/data-variables`.

```
[student@workstation ~]$ cd ~/data-variables  
[student@workstation data-variables]$
```

- 2. In den nächsten Schritten erstellen Sie ein Playbook, das den Apache-Webserver installiert und die Ports öffnet, damit der Service erreichbar ist. Das Playbook fragt den Webserver ab, um sicherzustellen, dass er gestartet wurde und ausgeführt wird.

Erstellen Sie das Playbook `playbook.yml`, und definieren Sie die folgenden Variablen im Abschnitt `vars`:

## Variablen

| Variable         | Beschreibung                                        |
|------------------|-----------------------------------------------------|
| web_pkg          | Zu installierendes Webserverpaket                   |
| firewall_pkg     | Zu installierendes Firewall-Paket                   |
| web_service      | Zu verwaltender Web-Service                         |
| firewall_service | Zu verwaltender Firewall-Service                    |
| python_pkg       | Erforderliches Paket für das Modul <code>uri</code> |
| rule             | Name des zu öffnenden Services                      |

```
---
- name: Deploy and start Apache HTTPD service
  hosts: webserver
  vars:
    web_pkg: httpd
    firewall_pkg: firewalld
    web_service: httpd
    firewall_service: firewalld
    python_pkg: python3-PyMySQL
    rule: http
```

- 3. Erstellen Sie den Block tasks und die erste Aufgabe, die mithilfe des Moduls `yum` sicherstellen sollte, dass die neuesten Versionen der erforderlichen Pakete installiert sind.

```
tasks:
- name: Required packages are installed and up to date
  yum:
    name:
      - "{{ web_pkg }}"
      - "{{ firewall_pkg }}"
      - "{{ python_pkg }}"
    state: latest
```



### Anmerkung

Sie können `ansible-doc yum` verwenden, um die Syntax für das Modul `yum` zu überprüfen. Die Syntax zeigt, dass die zugehörige Anweisung `name` eine Liste von Paketen enthalten kann, mit denen das Modul arbeiten sollte, damit Sie die Aufgaben nicht trennen müssen, um sicherzustellen, dass die einzelnen Pakete auf dem neuesten Stand sind.

- 4. Erstellen Sie zwei Aufgaben, um sicherzustellen, dass die Services `httpd` und `firewalld` gestartet und aktiviert werden.

```
- name: The {{ firewall_service }} service is started and enabled
  service:
    name: "{{ firewall_service }}"
    enabled: true
    state: started

- name: The {{ web_service }} service is started and enabled
  service:
    name: "{{ web_service }}"
    enabled: true
    state: started
```



### Anmerkung

Wie in `ansible-doc service` dokumentiert, arbeitet das Modul `service` anders als das Modul `yum`. Seine `name`-Anweisung enthält den Namen von genau einem Service, mit dem gearbeitet werden soll.

Sie können eine einzelne Aufgabe schreiben, die sicherstellt, dass beide Services gestartet und aktiviert werden, wobei das Keyword `loop` zum Einsatz kommt, das später in diesem Kurs behandelt wird.

- 5. Fügen Sie eine Aufgabe hinzu, die sicherstellt, dass bestimmte Inhalte in der Datei `/var/www/html/index.html` vorhanden sind.

```
- name: Web content is in place
  copy:
    content: "Example web content"
    dest: /var/www/html/index.html
```

- 6. Fügen Sie eine Aufgabe hinzu, die das Modul `firewalld` verwendet, um sicherzustellen, dass die Firewall-Ports für den in der Variablen `rule` benannten Service `firewalld` offen sind.

```
- name: The firewall port for {{ rule }} is open
  firewalld:
    service: "{{ rule }}"
    permanent: true
    immediate: true
    state: enabled
```

- 7. Erstellen Sie ein neues Play, das den Webservice abfragt, um sicherzustellen, dass alles ordnungsgemäß konfiguriert wurde. Das Play sollte auf `localhost` ausgeführt werden. Daher muss Ansible die Identität nicht wechseln, also legen Sie das Modul `become` auf `false` fest. Mit dem Modul `uri` kann eine URL überprüft werden. Überprüfen Sie für diese Aufgabe, ob der Statuscode 200 zurückgegeben wird, um zu bestätigen, dass der

Webserver auf `servera.lab.example.com` ausgeführt wird und ordnungsgemäß konfiguriert ist.

```
- name: Verify the Apache service
hosts: localhost
become: false
tasks:
  - name: Ensure the webserver is reachable
    uri:
      url: http://servera.lab.example.com
      status_code: 200
```

- 8. Wenn Sie fertig sind, sollte das Playbook wie folgt aussehen: Überprüfen Sie das Playbook und bestätigen Sie, dass beide Plays korrekt sind.

```
---
- name: Deploy and start Apache HTTPD service
hosts: webserver
vars:
  web_pkg: httpd
  firewall_pkg: firewalld
  web_service: httpd
  firewall_service: firewalld
  python_pkg: python3-PyMySQL
  rule: http

tasks:
  - name: Required packages are installed and up to date
    yum:
      name:
        - "{{ web_pkg }}"
        - "{{ firewall_pkg }}"
        - "{{ python_pkg }}"
      state: latest

  - name: The {{ firewall_service }} service is started and enabled
    service:
      name: "{{ firewall_service }}"
      enabled: true
      state: started

  - name: The {{ web_service }} service is started and enabled
    service:
      name: "{{ web_service }}"
      enabled: true
      state: started

  - name: Web content is in place
    copy:
      content: "Example web content"
      dest: /var/www/html/index.html

  - name: The firewall port for {{ rule }} is open
    firewalld:
```

```
service: "{{ rule }}"
permanent: true
immediate: true
state: enabled

- name: Verify the Apache service
hosts: localhost
become: false
tasks:
  - name: Ensure the webserver is reachable
    uri:
      url: http://servera.lab.example.com
      status_code: 200
```

- ▶ 9. Führen Sie vor dem Ausführen des Playbooks den Befehl `ansible-playbook --syntax-check` aus, um die zugehörige Syntax zu verifizieren. Wenn Fehler gemeldet werden, korrigieren Sie sie, bevor Sie mit dem nächsten Schritt fortfahren. Es sollte eine Ausgabe ähnlich der folgenden angezeigt werden:

```
[student@workstation data-variables]$ ansible-playbook --syntax-check playbook.yml
playbook: playbook.yml
```

- ▶ 10. Führen Sie mit dem Befehl `ansible-playbook` das Playbook aus. Beachten Sie die Ausgabe, während Ansible die Pakete installiert, die Services startet und aktiviert und sicherstellt, dass der Zugriff auf den Webserver möglich ist.

```
[student@workstation data-variables]$ ansible-playbook playbook.yml
PLAY [Deploy and start Apache HTTPD service] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Required packages are installed and up to date] ****
changed: [servera.lab.example.com]

TASK [The firewalld service is started and enabled] ****
ok: [servera.lab.example.com]

TASK [The httpd service is started and enabled] ****
changed: [servera.lab.example.com]

TASK [Web content is in place] ****
changed: [servera.lab.example.com]

TASK [The firewall port for http is open] ****
changed: [servera.lab.example.com]

PLAY [Verify the Apache service] ****
TASK [Gathering Facts] ****
ok: [localhost]
```

```
TASK [Ensure the webserver is reachable] ****
ok: [localhost]

PLAY RECAP ****
localhost : ok=2    changed=0    unreachable=0   failed=0
servera.lab.example.com : ok=6    changed=4    unreachable=0   failed=0
```

## Beenden

Führen Sie auf workstation das Skript `lab data-variables finish` aus, um diese Übung zu bereinigen.

```
[student@workstation ~]$ lab data-variables finish
```

Hiermit ist die angeleitete Übung beendet.

# Verwalten von Secrets

## Ziele

In diesem Abschnitt werden das Verschlüsseln sensibler Variablen mit Ansible Vault und das Ausführen von Playbooks behandelt, die auf Vault-verschlüsselte Variablen dateien verweisen.

## Präsentation von Ansible Vault

Ansible benötigt möglicherweise Zugang zu sensiblen Daten, wie Passwörtern oder API-Schlüsseln, um verwaltete Hosts zu konfigurieren. In der Regel können diese Informationen im Klartext in Inventarvariablen oder anderen Ansible-Dateien gespeichert werden. In diesem Fall hat jeder Benutzer, der Zugriff auf die Ansible-Dateien oder ein Versionskontrollsystem hat, das die Ansible-Dateien speichert, jedoch auch Zugriff auf diese sensiblen Daten. Dies stellt ein Sicherheitsrisiko dar.

Das in Ansible integrierte Ansible Vault kann zum Ver- und Entschlüsseln der von Ansible verwendeten strukturierten Datendateien verwendet werden. Um Ansible Vault nutzen zu können, werden mithilfe des Befehlszeilentools `ansible-vault` Dateien erstellt, bearbeitet, verschlüsselt, entschlüsselt und angezeigt. Ansible Vault kann alle strukturierten Datendateien, die von Ansible verwendet werden, verschlüsseln. Dazu gehören u. a. Inventarvariablen, in Playbooks enthaltene Variablen dateien, Variablen dateien, die bei Ausführung des Playbooks als Argument weitergegeben werden, oder Variablen, die in Ansible-Rollen definiert sind.



### Wichtig

Ansible Vault implementiert nicht seine eigenen kryptografischen Funktionen, sondern verwendet ein externes Python-Toolkit. Dateien werden durch eine symmetrische Verschlüsselung mit AES256 und einem Passwort als geheimem Schlüssel geschützt. Beachten Sie, dass diese Vorgehensweise von einem Drittanbieter nicht offiziell geprüft wurde.

## Erstellen einer verschlüsselten Datei

Führen Sie zum Erstellen einer neuen verschlüsselten Datei den Befehl `ansible-vault create filename` aus. Der Befehl fragt das neue Vault-Passwort ab und öffnet eine Datei mithilfe des Standardeditors `vi`. Sie können die Umgebungsvariable `EDITOR` so festlegen und exportieren, dass durch die Einstellung und den Export ein anderer Standardeditor angegeben wird. Beispiel: Um den Standardeditor auf `nano` festzulegen, verwenden Sie `export EDITOR=nano`.

```
[student@demo ~]$ ansible-vault create secret.yml
New Vault password: redhat
Confirm New Vault password: redhat
```

Anstatt das Vault-Passwort über die Standardeingabe einzugeben, können Sie eine Vault-Password datei verwenden, um das Vault-Passwort zu speichern. Sie müssen diese Datei mit Dateiberechtigungen und anderen Methoden sorgfältig schützen.

```
[student@demo ~]$ ansible-vault create --vault-password-file=vault-pass secret.yml
```

Als Verschlüsselungsmethode für den Schutz von Dateien wurde in den neueren Ansible-Versionen AES256 verwendet. Für Dateien, die mit älteren Versionen verschlüsselt wurden, wird u. U. weiterhin 128-Bit AES verwendet.

## Anzeigen einer verschlüsselten Datei

Sie können den Befehl `ansible-vault view filename` ausführen, um eine Ansible Vault-verschlüsselte Datei anzuzeigen, ohne sie für die Bearbeitung zu öffnen.

```
[student@demo ~]$ ansible-vault view secret1.yml
Vault password: secret
my_secret: "yJJvPqhsiusmmmPPZdnjndkdnYNDjdj782meUZcw"
```

## Bearbeiten einer vorhandenen verschlüsselten Datei

Ansible Vault bietet zur Bearbeitung einer vorhandenen, verschlüsselten Datei den Befehl `ansible-vault edit filename`. Dieser Befehl entschlüsselt die Datei als temporäre Datei und ermöglicht somit deren Bearbeitung. Nach dem Speichern werden die Inhalte kopiert und die temporäre Datei entfernt.

```
[student@demo ~]$ ansible-vault edit secret.yml
Vault password: redhat
```



### Anmerkung

Mit dem Sub-Befehl `edit` wird die Datei neu erstellt. Dieser sollte daher nur ausgeführt werden, wenn Änderungen vorgenommen werden. Dies kann Beeinträchtigungen mit sich bringen, wenn die Datei der Versionskontrolle unterliegt. Der Sub-Befehl `view` sollte immer dann verwendet werden, wenn die Dateiinhalte angezeigt werden sollen, ohne Änderungen vorzunehmen.

## Verschlüsseln einer vorhandenen Datei

Um eine bereits vorhandene Datei zu verschlüsseln, führen Sie den Befehl `ansible-vault encrypt filename` aus. Dieser Befehl kann die Namen mehrerer Dateien als Argumente verschlüsseln.

```
[student@demo ~]$ ansible-vault encrypt secret1.yml secret2.yml
New Vault password: redhat
Confirm New Vault password: redhat
Encryption successful
```

Verwenden Sie die Option `--output=OUTPUT_FILE`, um die verschlüsselte Datei mit einem neuen Namen zu speichern. Sie können nur eine Eingabedatei mit der Option `--output` verwenden.

## Entschlüsseln einer vorhandenen Datei

Eine vorhandene, verschlüsselte Datei kann mit dem Befehl `ansible-vault decrypt filename` dauerhaft entschlüsselt werden. Sie können die Option `--output` verwenden, wenn

eine einzelne Datei entschlüsselt werden soll, um die entschlüsselte Datei unter einem anderen Namen zu speichern.

```
[student@demo ~]$ ansible-vault decrypt secret1.yml --output=secret1-decrypted.yml  
Vault password: redhat  
Decryption successful
```

## Ändern des Passworts einer verschlüsselten Datei

Sie können den Befehl `ansible-vault rekey filename` ausführen, um das Passwort einer verschlüsselten Datei zu ändern. Dieser Befehl kann für mehrere Datendateien gleichzeitig einen neuen Schlüssel erstellen. Er fordert auf, das ursprüngliche Passwort und das neue Passwort einzugeben.

```
[student@demo ~]$ ansible-vault rekey secret.yml  
Vault password: redhat  
New Vault password: RedHat  
Confirm New Vault password: RedHat  
Rekey successful
```

Verwenden Sie bei Vault-Passworddateien die Option `--new-vault-password-file`:

```
[student@demo ~]$ ansible-vault rekey \  
> --new-vault-password-file=NEW_VAULT_PASSWORD_FILE secret.yml
```

## Playbooks und Ansible Vault

Zur Ausführung eines Playbooks, das auf Dateien zugreift, die mit Ansible Vault verschlüsselt wurden, müssen Sie das Verschlüsselungspasswort für den Befehl `ansible-playbook` bereitstellen. Wenn Sie das Passwort nicht angeben, meldet das Playbook einen Fehler:

```
[student@demo ~]$ ansible-playbook site.yml  
ERROR: A vault password must be specified to decrypt vars/api_key.yml
```

Verwenden Sie die Option `--vault-id`, um das Vault-Passwort für das Playbook anzugeben. Wenn Sie beispielsweise das Vault-Passwort interaktiv angeben möchten, verwenden Sie `--vault-id @prompt`, wie im folgenden Beispiel dargestellt:

```
[student@demo ~]$ ansible-playbook --vault-id @prompt site.yml  
Vault password (default): redhat
```



### Wichtig

Wenn Sie eine Version von Ansible vor der Version 2.4 verwenden, müssen Sie die Option `--ask-vault-pass` verwenden, um das Vault-Passwort interaktiv bereitzustellen. Sie können diese Option weiterhin verwenden, wenn alle Vault-verschlüsselten Dateien, die vom Playbook verwendet wurden, mit demselben Passwort verschlüsselt wurden.

```
[student@demo ~]$ ansible-playbook --ask-vault-pass site.yml  
Vault password: redhat
```

Alternativ können Sie die Option `--vault-password-file` verwenden, um eine Datei anzugeben, die das Verschlüsselungspasswort in Klartext speichert. Das Passwort sollte in der Datei als Zeichenfolge in einer einzelnen Zeile gespeichert werden. Da die Datei ein Passwort im Klartext enthält, ist es zwingend erforderlich, das Passwort durch Dateiberechtigungen und andere Sicherheitsmaßnahmen zu schützen.

```
[student@demo ~]$ ansible-playbook --vault-password-file=vault-pw-file site.yml
```

Sie können auch die Umgebungsvariable `ANSIBLE_VAULT_PASSWORD_FILE` verwenden, um den Standardspeicherort der Passworddatei anzugeben.



### Wichtig

Ab Ansible 2.4 können Sie mehrere Ansible Vault-Passwörter mit `ansible-playbook` verwenden. Geben Sie zum Verwenden mehrerer Passwörter mehrere `--vault-id`- oder `--vault-password-file`-Optionen an den Befehl `ansible-playbook` weiter.

```
[student@demo ~]$ ansible-playbook \  
> --vault-id one@prompt --vault-id two@prompt site.yml  
Vault password (one):  
Vault password (two):  
...output omitted...
```

Die Vault-IDs `one` und `two`, die `@prompt` vorangehen, können beliebige Werte sein, die Sie sogar vollständig auslassen können. Wenn Sie jedoch die Option `--vault-id id` beim Verschlüsseln einer Datei mit dem Befehl `ansible-vault` verwenden, wenn Sie `ansible-playbook` ausführen, wird das Passwort für die übereinstimmende ID vor allen anderen versucht. Falls es nicht übereinstimmt, werden die anderen von Ihnen bereitgestellten Passwörter als Nächstes versucht. Die Vault-ID `@prompt` ohne ID ist eigentlich die Kurzform für `default@prompt`, d. h., es wird zur Eingabe des Passworts für die Vault-ID `default` aufgefordert.

## Empfohlene Verfahren für die Verwaltung von Variabldateien

Zur Vereinfachung der Verwaltung ist es sinnvoll, Ansible-Projekte so einzurichten, dass sensible Variablen und alle anderen Variablen in separaten Dateien aufbewahrt werden. Die Dateien, die sensible Variablen enthalten, können mithilfe des Befehls `ansible-vault` geschützt werden.

Beachten Sie, dass zur Verwaltung von Gruppen- und Hostvariablen nach Möglichkeit Verzeichnisse auf Playbook-Ebene erstellt werden sollten. Das Verzeichnis `group_vars` enthält in der Regel Variabldateien mit Namen, die den Hostgruppen entsprechen, auf die sie angewendet werden. Das Verzeichnis `host_vars` enthält in der Regel Variabldateien mit Namen, die den Hostnamen der verwalteten Hosts entsprechen, auf die sie angewendet werden.

Anstatt Dateien in `group_vars` oder `host_vars` zu nutzen, können Sie für die jeweiligen Hostgruppen bzw. verwalteten Hosts auch Verzeichnisse verwenden. Diese Verzeichnisse können mehrere Variabldateien enthalten, die alle von der Hostgruppe bzw. dem verwalteten Host verwendet werden. Beispielsweise verwenden im folgenden Projektverzeichnis für `playbook.yml` Mitglieder der Hostgruppe `webservers` Variablen aus der Datei `group_vars/webservers/vars`. Der Host `demo.example.com` verwendet die Variablen aus den beiden Dateien `host_vars/demo.example.com/vars` und `host_vars/demo.example.com/vault`:

```
.
├── ansible.cfg
├── group_vars
│   └── webservers
│       └── vars
├── host_vars
│   └── demo.example.com
│       ├── vars
│       └── vault
└── inventory
└── playbook.yml
```

In diesem Szenario liegt der Vorteil darin, dass die meisten Variablen für `demo.example.com` in der Datei `vars` platziert werden können. Demgegenüber können sensible Variablen geheim gehalten werden, indem sie separat in der Datei `vault` platziert werden. Der Administrator kann anschließend mit dem Befehl `ansible-vault` die Datei `vault` verschlüsseln, wobei die Datei `vars` im Klartext belassen wird.

Die in diesem Beispiel im Verzeichnis `host_vars/demo.example.com` verwendeten Dateinamen sind unspezifisch. Dieses Verzeichnis könnte weitere, mit Ansible Vault verschlüsselte wie auch unverschlüsselte Dateien enthalten.

Playbook-Variablen (im Gegensatz zu Inventarvariablen) können ebenfalls mit Ansible Vault geschützt werden. Sensible Playbook-Variablen können in einer separaten Datei platziert werden, die mit Ansible Vault verschlüsselt und mittels der Anweisung `vars_files` zum Playbook hinzugefügt wird. Dies kann hilfreich sein, da Playbook-Variablen Vorrang vor Inventardateien haben.

Wenn Sie mehrere Vault-Passwörter für Ihr Playbook verwenden, stellen Sie sicher, dass jeder verschlüsselten Datei eine Vault-ID zugewiesen ist und dass Sie beim Ausführen des Playbooks das mit der Vault-ID übereinstimmende Passwort eingeben. Dadurch wird sichergestellt, dass das richtige Passwort zuerst ausgewählt wird, wenn die Vault-verschlüsselte Datei entschlüsselt wird, was wiederum schneller ist, als Ansible zu zwingen, alle von Ihnen bereitgestellten Vault-Passwörter zu versuchen, bis das richtige gefunden wird.



### Literaturhinweise

Manpages `ansible-playbook(1)` und `ansible-vault(1)`

#### **Encrypting content with Ansible Vault – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/user\\_guide/vault.html](https://docs.ansible.com/ansible/2.9/user_guide/vault.html)

#### **Keep vaulted variables safely visible – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_best\\_practices.html#keep-vaulted-variables-safely-visible](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_best_practices.html#keep-vaulted-variables-safely-visible)

## ► Angeleitete Übung

# Verwalten von Secrets

In dieser Übung verschlüsseln Sie sensible Variablen mit Ansible Vault, um sie zu schützen, und führen anschließend ein Playbook aus, das diese Variablen verwendet.

## Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Führen Sie ein Playbook unter Verwendung der in einer verschlüsselten Datei definierten Variablen aus.

## Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab data-secret start` aus. Dieses Skript stellt sicher, dass Ansible auf `workstation` installiert ist, und erstellt ein Arbeitsverzeichnis für diese Übung. Dieses Verzeichnis enthält eine Inventardatei, die auf `servera.lab.example.com` als verwalteten Host verweist, der Teil der Gruppe `devservers` ist.

```
[student@workstation ~]$ lab data-secret start
```

## Anweisungen

- 1. Wechseln Sie auf `workstation` als Benutzer `student` ins Arbeitsverzeichnis `/home/student/data-secret`.

```
[student@workstation ~]$ cd ~/data-secret  
[student@workstation data-secret]$
```

- 2. Bearbeiten Sie den Inhalt der bereitgestellten verschlüsselten Datei `secret.yml`. Die Datei kann mit dem Passwort `redhat` entschlüsselt werden. Kommentieren Sie die Variableneinträge `username` und `pwhash` aus.

- 2.1. Bearbeiten Sie die verschlüsselte Datei `/home/student/data-secret/secret.yml`. Geben Sie, wenn Sie dazu aufgefordert werden, das Passwort `redhat` für den Vault an. Die verschlüsselte Datei wird im Standardeditor `vim` geöffnet.

```
[student@workstation data-secret]$ ansible-vault edit secret.yml  
Vault password: redhat
```

- 2.2. Heben Sie die Auskommentierung der beiden Variableneinträge auf, speichern Sie die Datei und beenden Sie den Editor. Sie sollten wie folgt aussehen:

```
username: ansibleuser1
pwhash: $6$jf...uxhP1
```

- 3. Erstellen Sie ein Playbook mit dem Namen /home/student/data-secret/create\_users.yml, das die in der verschlüsselten Datei /home/student/data-secret/secret.yml definierten Variablen verwendet.

Konfigurieren Sie das Playbook für die Verwendung der Hostgruppe devservers.

Führen Sie dieses Playbook als devops-Benutzer auf dem verwalteten Remote-Host aus.

Konfigurieren Sie das Playbook, um den von der Variablen username definierten Benutzer ansibleuser1 zu erstellen. Legen Sie mithilfe des in der Variablen pwhash gespeicherten Passwort-Hash-Werts das Passwort des Benutzers fest.

```
---
- name: create user accounts for all our servers
  hosts: devservers
  become: True
  remote_user: devops
  vars_files:
    - secret.yml
  tasks:
    - name: Creating user from secret.yml
      user:
        name: "{{ username }}"
        password: "{{ pwhash }}"
```

- 4. Führen Sie den Befehl ansible-playbook --syntax-check aus, um die Syntax des Playbooks create\_users.yml zu überprüfen. Verwenden Sie die Option --ask-vault-pass, damit zur Eingabe des Vault-Passworts aufgefordert wird, mit dem die Datei secret.yml verschlüsselt wird. Beheben Sie alle Syntaxfehler, bevor Sie fortfahren.

```
[student@workstation data-secret]$ ansible-playbook --syntax-check \
> --ask-vault-pass create_users.yml
Vault password (default): redhat

playbook: create_users.yml
```



### Anmerkung

Anstatt --ask-vault-pass zu verwenden, können Sie die neuere Option --vault-id @prompt nutzen, um dasselbe zu erledigen.

- 5. Erstellen Sie eine Passworddatei namens vault-pass, die anstelle der Passwordeingabe zur Ausführung des Playbooks verwendet werden soll. Die Datei muss den Klartext redhat als Vault-Passwort enthalten. Ändern Sie die Berechtigungen der Datei zu 0600.

```
[student@workstation data-secret]$ echo 'redhat' > vault-pass
[student@workstation data-secret]$ chmod 0600 vault-pass
```

- 6. Führen Sie das Ansible-Playbook mit der Datei `vault-pass` aus, um den Benutzer `ansibleuser1` auf einem Remote-System mithilfe der Passwörter zu erstellen, die als Variablen in der verschlüsselten Ansible Vault-Datei `secret.yml` gespeichert sind.

```
[student@workstation data-secret]$ ansible-playbook \
> --vault-password-file=vault-pass create_users.yml

PLAY [create user accounts for all our servers] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Creating users from secret.yml] ****
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```

- 7. Verifizieren Sie, dass das Playbook ordnungsgemäß ausgeführt wurde. Der Benutzer `ansibleuser1` sollte auf `servera.lab.example.com` vorhanden sein und über das richtige Passwort verfügen. Testen Sie dies, indem Sie `ssh` verwenden, um sich als dieser Benutzer auf `servera.lab.example.com` anzumelden. Das Passwort für `ansibleuser1` lautet `redhat`. Verwenden Sie beim Anmelden die Option `-o PreferredAuthentications=password`, um sicherzustellen, dass SSH nur versucht, die Authentifizierung nach Passwort und nicht nach SSH-Schlüssel vorzunehmen.

Melden Sie sich von `servera` ab, wenn Sie sich erfolgreich angemeldet haben.

```
[student@workstation data-secret]$ ssh -o PreferredAuthentications=password \
> ansibleuser1@servera.lab.example.com
ansibleuser1@servera.lab.example.com's password: redhat
Activate the web console with: systemctl enable --now cockpit.socket

[ansibleuser1@servera ~]$ exit
logout
Connection to servera.lab.example.com closed.
```

## Beenden

Führen Sie auf `workstation` das Skript `lab data-secret finish` aus, um diese Übung zu bereinigen.

```
[student@workstation ~]$ lab data-secret finish
```

Hiermit ist die angeleitete Übung beendet.

# Verwalten von Fakten

## Ziele

In diesem Abschnitt wird behandelt, wie mithilfe von Ansible-Fakten auf Daten über verwaltete Hosts verwiesen wird, und wie benutzerdefinierte Fakten über verwaltete Hosts konfiguriert werden.

## Beschreiben von Ansible-Fakten

Ansible-Fakten sind Variablen, die von Ansible automatisch auf einem verwalteten Host ermittelt werden. Fakten enthalten hostspezifische Funktionen, die wie normale Variablen in Plays, Bedingungen, Loops oder anderen Anweisungen verwendet werden können, die von einem Wert abhängen, der von einem verwalteten Host erfasst wird.

Die für einen verwalteten Host erfassten Fakten beinhalten unter anderem:

- Den Hostnamen
- Die Kernel-Version
- Die Netzwerkschnittstellen
- Die IP-Adressen
- Die Version des Betriebssystems
- Verschiedene Umgebungsvariablen
- Die Anzahl der CPUs
- Den verfügbaren oder freien Arbeitsspeicher
- Den verfügbaren Festplattenspeicher

Fakten bieten eine praktische Möglichkeit, den Status eines verwalteten Knotens abzurufen und zu entscheiden, welche Aktion aufgrund dieses Status ausgeführt werden soll. Beispiel:

- Ein Server kann durch eine bedingte Aufgabe neu gestartet werden, deren Ausführung von einem Fakt abhängt, das die aktuelle Kernel-Version des verwalteten Hosts enthält.
- Die MySQL-Konfigurationsdatei kann in Abhängigkeit von dem verfügbaren Speicherplatz angepasst werden, der von einem Fakt gemeldet wird.
- Die in einer Konfigurationsdatei verwendete IPv4-Adresse kann abhängig von dem Wert eines Fakts festgelegt werden.

Normalerweise führt jedes Play vor der ersten Aufgabe automatisch das Modul `setup` aus, um Fakten zu sammeln. Dies wird in Ansible 2.3 und höher als Aufgabe `Gathering Facts` und in älteren Versionen von Ansible einfach als `setup` gemeldet. Standardmäßig benötigen Sie keine Aufgabe, um `setup` in Ihrem Play auszuführen. Normalerweise wird es automatisch für Sie ausgeführt.

### Kapitel 3 | Verwalten von Variablen und Fakten

Um herauszufinden, welche Fakten für Ihre verwalteten Hosts gesammelt werden, können Sie ein kurzes Playbook erstellen, das Fakten sammelt und das Modul `debug` verwendet, um den Wert der Variablen `ansible_facts` auszugeben.

```
- name: Fact dump
hosts: all
tasks:
  - name: Print all facts
    debug:
      var: ansible_facts
```

Wenn Sie das Playbook ausführen, werden die Fakten in der Jobausgabe angezeigt:

```
[user@demo ~]$ ansible-playbook facts.yml

PLAY [Fact dump] ****
TASK [Gathering Facts] ****
ok: [demo1.example.com]

TASK [Print all facts] ****
ok: [demo1.example.com] => {
  "ansible_facts": {
    "all_ipv4_addresses": [
      "172.25.250.10"
    ],
    "all_ipv6_addresses": [
      "fe80::5054:ff:fe00:fa0a"
    ],
    "ansible_local": {},
    "apparmor": {
      "status": "disabled"
    },
    "architecture": "x86_64",
    "bios_date": "01/01/2011",
    "bios_version": "0.5.1",
    "cmdline": {
      "BOOT_IMAGE": "/boot/vmlinuz-3.10.0-327.el7.x86_64",
      "LANG": "en_US.UTF-8",
      "console": "ttyS0,115200n8",
      "crashkernel": "auto",
      "net.ifnames": "0",
      "no_timer_check": true,
      "ro": true,
      "root": "UUID=2460ab6e-e869-4011-acae-31b2e8c05a3b"
    },
    ...output omitted...
  }
}
```

Das Playbook zeigt den Inhalt der Variable `ansible_facts` im JSON-Format als Hash/Dictionary von Variablen. Sie können die Ausgabe durchsuchen, um festzustellen, welche Fakten gesammelt werden, um Fakten zu erhalten, die sie evtl. in Ihren Plays verwenden möchten.

Die folgende Tabelle enthält einige Fakten, die auf einem verwalteten Knoten gesammelt werden könnten und in einem Playbook nützlich sein können:

## Beispiele für Ansible-Fakten

| Fakt                                         | Variable                                                                   |
|----------------------------------------------|----------------------------------------------------------------------------|
| Kurzer Hostname                              | <code>ansible_facts['hostname']</code>                                     |
| Vollqualifizierter Domain Name               | <code>ansible_facts['fqdn']</code>                                         |
| IPv4-Hauptadresse (basierend auf Routing)    | <code>ansible_facts['default_ipv4']['address']</code>                      |
| Liste der Namen aller Netzwerkschnittstellen | <code>ansible_facts['interfaces']</code>                                   |
| Größe der Festplattenpartition /dev/vda1     | <code>ansible_facts['devices']['vda']['partitions']['vda1']['size']</code> |
| Liste von DNS-Servern                        | <code>ansible_facts['dns']['nameservers']</code>                           |
| Version des aktuell ausgeführten Kernels     | <code>ansible_facts['kernel']</code>                                       |



### Anmerkung

Denken Sie daran, dass es, wenn der Wert einer Variablen ein Hash/Dictionary ist, zwei Syntaxen gibt, die zum Abrufen des Wertes verwendet werden können. Zwei Beispiele aus der vorhergehenden Tabelle:

- `ansible_facts['default_ipv4']['address']` kann auch als `ansible_facts.default_ipv4.address` geschrieben werden.
- `ansible_facts['dns']['nameservers']` kann auch als `ansible_facts.dns.nameservers` geschrieben werden.

Wenn ein Fakt in einem Playbook verwendet wird, ersetzt Ansible dynamisch den Variablenamen für den Fakt durch den entsprechenden Wert:

```
---
- hosts: all
  tasks:
    - name: Prints various Ansible facts
      debug:
        msg: >
          The default IPv4 address of {{ ansible_facts.fqdn }}
          is {{ ansible_facts.default_ipv4.address }}
```

Die folgende Ausgabe zeigt, wie Ansible den verwalteten Knoten abfragt und die Variable dynamisch mit den Systeminformationen aktualisiert. Mit Fakten können auch dynamische Gruppen von Hosts erstellt werden, die bestimmten Kriterien entsprechen.

```
[user@demo ~]$ ansible-playbook playbook.yml
PLAY ****
```

```

TASK [Gathering Facts] *****
ok: [demo1.example.com]

TASK [Prints various Ansible facts] *****
ok: [demo1.example.com] => {
    "msg": "The default IPv4 address of demo1.example.com is
           172.25.250.10"
}

PLAY RECAP *****
demo1.example.com      : ok=2      changed=0      unreachable=0      failed=0

```

## Als Variablen injizierte Ansible-Fakten

Vor Ansible 2.5 wurden Fakten als einzelne Variablen injiziert, denen die Zeichenfolge `ansible_` vorangestellt war, anstatt Teil der Variablen `ansible_facts` zu sein. Zum Beispiel wäre der Fakt `ansible_facts['distribution']` als `ansible_distribution` bezeichnet worden.

In vielen älteren Playbooks werden anstelle der neuen Syntax, die unter der Variablen `ansible_facts` einen Namespace besitzt, weiterhin als Variablen injizierte Fakten verwendet. Sie können einen Ad-hoc-Befehl verwenden, um das Modul `setup` auszuführen, damit der Wert sämtlicher Fakten in dieser Form ausgegeben wird. Im folgenden Beispiel wird ein Ad-hoc-Befehl verwendet, um das Modul `setup` auf dem verwalteten Host `demo1.example.com` auszuführen:

```

[user@demo ~]$ ansible demo1.example.com -m setup
demo1.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "172.25.250.10"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::5054:ff:fe00:fa0a"
        ],
        "ansible_apparmor": {
            "status": "disabled"
        },
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "01/01/2011",
        "ansible_bios_version": "0.5.1",
        "ansible_cmdline": {
            "BOOT_IMAGE": "/boot/vmlinuz-3.10.0-327.el7.x86_64",
            "LANG": "en_US.UTF-8",
            "console": "ttyS0,115200n8",
            "crashkernel": "auto",
            "net.ifnames": "0",
            "no_timer_check": true,
            "ro": true,
            "root": "UUID=2460ab6e-e869-4011-acae-31b2e8c05a3b"
        }
    ...output omitted...
}

```

In der folgenden Tabelle werden die alten und neuen Faktennamen verglichen.

## Vergleich ausgewählter Ansible-Faktennamen

| ansible_facts-Form                                            | Alte Faktenvariablenform                             |
|---------------------------------------------------------------|------------------------------------------------------|
| ansible_facts['hostname']                                     | ansible_hostname                                     |
| ansible_facts['fqdn']                                         | ansible_fqdn                                         |
| ansible_facts['default_ipv4']['address']                      | ansible_default_ipv4['address']                      |
| ansible_facts['interfaces']                                   | ansible_interfaces                                   |
| ansible_facts['devices']['vda']['partitions']['vda1']['size'] | ansible_devices['vda']['partitions']['vda1']['size'] |
| ansible_facts['dns']['nameservers']                           | ansible_dns['nameservers']                           |
| ansible_facts['kernel']                                       | ansible_kernel                                       |



### Wichtig

Derzeit erkennt Ansible das neue Faktenbenennungssystem (mit `ansible_facts`) und das alte Benennungssystem „Als separate Variablen injizierte Fakten“ der Versionen vor 2.5.

Sie können das alte Benennungssystem ausschalten, indem Sie den Parameter `inject_facts_as_vars` im Abschnitt `[default]` der Ansible-Konfigurationsdatei auf `false` festlegen. Die aktuelle Standardeinstellung lautet `true`.

Der Standardwert von `inject_facts_as_vars` wird in einer künftigen Version von Ansible wahrscheinlich in `false` geändert. Wenn der Wert auf `false` eingestellt ist, können Sie nur mithilfe des neuen `ansible_facts.*`-Benennungssystems auf Ansible-Fakten verweisen. Steven

```
...output omitted...
TASK [Show me the facts] *****
fatal: [demo.example.com]: FAILED! => {"msg": "The task includes an option
with an undefined variable. The error was: 'ansible_distribution' is
undefined\n\nThe error appears to have been in
'/home/student/demo/playbook.yml': line 5, column 7, but may\nbe elsewhere in
the file depending on the exact syntax problem.\n\nThe offending line appears
to be:\n\n  tasks:\n    - name: Show me the facts\n          ^ here\n"}
...output omitted...
```

## Deaktivieren der Faktensammlung

Manchmal möchten Sie keine Fakten für Ihr Play sammeln. Es gibt einige Gründe, warum dies der Fall sein könnte. Vielleicht verwenden Sie keine Fakten und möchten das Play beschleunigen oder die Rechenlast reduzieren, die vom Play auf den verwalteten Hosts verursacht wird. Vielleicht können die verwalteten Hosts das Modul `setup` aus irgendeinem Grund nicht ausführen oder es muss Software als Voraussetzung für das Sammeln von Fakten installiert werden.

### Kapitel 3 | Verwalten von Variablen und Fakten

Um die Faktensammlung für ein Play zu deaktivieren, setzen Sie den Schlüssel `gather_facts` auf `no`:

```
---
```

```
- name: This play gathers no facts automatically
  hosts: large_farm
  gather_facts: no
```

Auch wenn `gather_facts: no` für ein Play festgelegt ist, können jederzeit manuell Fakten gesammelt werden, indem eine Aufgabe ausgeführt wird, die das Modul `setup` verwendet:

```
tasks:
  - name: Manually gather facts
    setup:
      ...output omitted...
```

## Erstellen von benutzerdefinierten Fakten

Administratoren können *benutzerdefinierte Fakten* erstellen, die auf dem jeweiligen verwalteten Host lokal gespeichert werden. Diese Fakten werden in die Liste der Standardfakten integriert, die vom Modul `setup` gesammelt werden, wenn es auf dem verwalteten Host ausgeführt wird. Diese ermöglichen es dem verwalteten Host, Ansible beliebige Variablen zur Verfügung zu stellen, mit denen das Verhalten von Plays angepasst werden kann.

Benutzerdefinierte Fakten können in einer statischen Datei definiert werden, die als INI-Datei oder mithilfe von JSON formatiert ist. Sie können auch in Form ausführbarer Skripte vorliegen, die eine JSON-Ausgabe generieren.

Benutzerdefinierte Fakten ermöglichen es Administratoren, bestimmte Werte für verwaltete Hosts zu definieren, die von Plays zum Füllen von Konfigurationsdateien oder zum bedingten Ausführen von Aufgaben verwendet werden können. Mithilfe von dynamischen benutzerdefinierten Fakten lassen sich die Werte für diese Fakten und sogar die Liste der Fakten, die bereitgestellt werden, programmgesteuert ermitteln, wenn das Play ausgeführt wird.

Standardmäßig lädt das Modul `setup` benutzerdefinierte Fakten aus Dateien und Skripten im Verzeichnis `/etc/ansible/facts.d` der einzelnen verwalteten Hosts. Die Namen von allen Dateien und Skripten müssen auf `.fact` enden, damit sie verwendet werden können. Skripte für dynamische benutzerdefinierte Fakten müssen JSON-formatierte Fakten ausgeben und ausführbar sein.

Dies ist ein Beispiel einer statischen Datei für benutzerdefinierte Fakten, die im INI-Format erstellt wurde. Eine INI-formatierte Datei für benutzerdefinierte Fakten enthält eine oberste Ebene, die durch einen Abschnitt, gefolgt von den Schlüssel-Wert-Paaren für die zu definierenden Fakten definiert wird:

```
[packages]
web_package = httpd
db_package = mariadb-server

[users]
user1 = joe
user2 = jane
```

Dieselben Fakten können auch im JSON-Format bereitgestellt werden. Die folgenden JSON-Fakten entsprechen den Fakten, die durch das INI-Format im vorherigen Beispiel angegeben werden. Die JSON-Daten können in einer statischen Textdatei gespeichert oder von einem ausführbaren Skript an die Standard-Ausgabe geleitet werden:

```
{  
    "packages": {  
        "web_package": "httpd",  
        "db_package": "mariadb-server"  
    },  
    "users": {  
        "user1": "joe",  
        "user2": "jane"  
    }  
}
```



### Anmerkung

Im Gegensatz zu Playbooks können Dateien für benutzerdefinierte Fakten das YAML-Format nicht verwenden. Das JSON-Format ist die nächstliegende Entsprechung.

Benutzerdefinierte Fakten werden vom Modul `setup` in der Variablen `ansible_facts['ansible_local']` gespeichert. Fakten werden basierend auf dem Namen der Datei organisiert, in der sie definiert werden. Nehmen wir etwa an, dass die voranstehenden benutzerdefinierten Fakten von einer Datei erzeugt werden, die als `/etc/ansible/facts.d/custom.fact` auf dem verwalteten Host gespeichert ist. Der Wert der Variable `ansible_facts['ansible_local']['custom']['users']['user1']` lautet in diesem Fall `joe`.

Sie können die Struktur Ihrer benutzerdefinierten Fakten prüfen, indem Sie das Modul `setup` mit einem Ad-hoc-Befehl auf den verwalteten Hosts ausführen.

```
[user@demo ~]$ ansible demo1.example.com -m setup  
demo1.example.com | SUCCESS => {  
    "ansible_facts": {  
        ...output omitted...  
        "ansible_local": {  
            "custom": {  
                "packages": {  
                    "db_package": "mariadb-server",  
                    "web_package": "httpd"  
                },  
                "users": {  
                    "user1": "joe",  
                    "user2": "jane"  
                }  
            },  
            ...output omitted...
        }
    }
}
...output omitted...
```

```
},
  "changed": false
}
```

Benutzerdefinierte Fakten können wie Standardfakten in Playbooks verwendet werden:

```
[user@demo ~]$ cat playbook.yml
---
- hosts: all
  tasks:
    - name: Prints various Ansible facts
      debug:
        msg: >
          The package to install on {{ ansible_facts['fqdn'] }}
          is {{ ansible_facts['ansible_local']['custom']['packages']
          ['web_package'] }}
```

```
[user@demo ~]$ ansible-playbook playbook.yml
PLAY ****
TASK [Gathering Facts] ****
ok: [demo1.example.com]

TASK [Prints various Ansible facts] ****
ok: [demo1.example.com] => {
    "msg": "The package to install on demo1.example.com  is httpd"
}

PLAY RECAP ****
demo1.example.com      : ok=2      changed=0      unreachable=0      failed=0
```

## Verwenden magischer Variablen

Manche Variablen sind weder Fakten noch werden sie durch das Modul `setup` konfiguriert, werden aber von Ansible ebenfalls automatisch festgelegt. Diese *magischen Variablen* können auch hilfreich sein, um spezifische Informationen für einen bestimmten verwalteten Host abzurufen.

Die folgenden vier sind besonders nützlich:

### `hostvars`

Enthält die Variablen für verwaltete Hosts und kann zum Abrufen der Werte von Variablen eines anderen verwalteten Hosts verwenden werden. Enthält nicht die Fakten für einen verwalteten Host, wenn sie für diesen Host noch nicht gesammelt wurden.

### `group_names`

Listet alle Gruppen auf, in denen der aktuelle verwaltete Host Mitglied ist.

### `groups`

Listet alle Gruppen und Hosts im Inventar auf.

### `inventory_hostname`

Enthält den Hostnamen für den aktuellen verwalteten Host, wie er im Inventar konfiguriert ist. Dieser kann sich aus verschiedenen Gründen von dem Hostnamen unterscheiden, der von Fakten gemeldet wird.

Es gibt noch eine Reihe weiterer „magischer Variablen“. Weitere Informationen finden Sie unter [https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_variables.html#variable-precedence-where-should-i-put-a-variable](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable). Eine Methode, Einblick in ihre Werte zu erhalten, besteht darin, das Modul `debug` so zu verwenden, dass der Inhalt der Variable `hostvars` für einen bestimmten Host gemeldet wird:

```
[user@demo ~]$ ansible localhost -m debug -a 'var=hostvars["localhost"]'
localhost | SUCCESS => {
    "hostvars[\"localhost\"]": {
        "ansible_check_mode": false,
        "ansible_connection": "local",
        "ansible_diff_mode": false,
        "ansible_facts": {},
        "ansibleforks": 5,
        "ansible_inventory_sources": [
            "/home/student/demo/inventory"
        ],
        "ansible_playbook_python": "/usr/bin/python2",
        "ansible_python_interpreter": "/usr/bin/python2",
        "ansible_verbosity": 0,
        "ansible_version": {
            "full": "2.7.0",
            "major": 2,
            "minor": 7,
            "revision": 0,
            "string": "2.7.0"
        },
        "group_names": [],
        "groups": {
            "all": [
                "serverb.lab.example.com"
            ],
            "ungrouped": [],
            "webservers": [
                "serverb.lab.example.com"
            ]
        },
        "inventory_hostname": "localhost",
        "inventory_hostname_short": "localhost",
        "omit": "__omit_place_holder__18d132963728b2cbf7143dd49dc4bf5745fe5ec3",
        "playbook_dir": "/home/student/demo"
    }
}
```



### Literaturhinweise

#### **setup – Gathers facts about remote hosts – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/modules/setup\\_module.html](https://docs.ansible.com/ansible/2.9/modules/setup_module.html)

#### **Variables discovered from systems: Facts – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_variables.html#variables-discovered-from-systems-facts](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_variables.html#variables-discovered-from-systems-facts)

## ► Angeleitete Übung

# Verwalten von Fakten

In dieser Übung sammeln Sie Ansible-Fakten von einem verwalteten Host und verwenden diese in Plays.

## Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Fakten von einem Host sammeln.
- Aufgaben erstellen, die gesammelte Fakten verwenden.

## Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab data-facts start` aus. Dieses Skript erstellt das Arbeitsverzeichnis `data-facts` und füllt es mit einer Ansible-Konfigurationsdatei und dem Hostinventar.

```
[student@workstation ~]$ lab data-facts start
```

## Anweisungen

- 1. Wechseln Sie auf `workstation` als Benutzer `student` in das Verzeichnis `/home/student/data-facts`.

```
[student@workstation ~]$ cd ~/data-facts  
[student@workstation data-facts]$
```

- 2. Das Ansible-Modul `setup` ruft die Fakten von Systemen ab. Führen Sie einen Ad-hoc-Befehl aus, um die Fakten für alle Server in der Gruppe `webserver` abzurufen. In der Ausgabe werden alle für `servera.lab.example.com` gesammelten Fakten im JSON-Format angezeigt. Überprüfen Sie einige der angezeigten Variablen.

```
[student@workstation data-facts]$ ansible webserver -m setup  
...output omitted...  
servera.lab.example.com | SUCCESS => {  
    "ansible_facts": {  
        "ansible_all_ipv4_addresses": [  
            "172.25.250.10"  
        ],  
        "ansible_all_ipv6_addresses": [  
            "fe80::2937:3aa3:ea8d:d3b1"  
        ],  
    ...output omitted...
```

- 3. Erstellen Sie auf `workstation` eine Faktendatei mit dem Namen/`home/student/data-facts/custom.fact`. Die Faktendatei definiert das zu installierende Paket und den auf `servera` zu startenden Service. Die Datei sollte wie folgt aussehen:

```
[general]
package = httpd
service = httpd
state = started
enabled = true
```

- 4. Erstellen Sie das Playbook `setup_facts.yml`, um das Remote-Verzeichnis `/etc/ansible/facts.d` zu erstellen und die Datei `custom.fact` in diesem Verzeichnis zu speichern.

```
---
- name: Install remote facts
  hosts: webserver
  vars:
    remote_dir: /etc/ansible/facts.d
    facts_file: custom.fact
  tasks:
    - name: Create the remote directory
      file:
        state: directory
        recurse: yes
        path: "{{ remote_dir }}"
    - name: Install the new facts
      copy:
        src: "{{ facts_file }}"
        dest: "{{ remote_dir }}"
```

- 5. Führen Sie einen Ad-hoc-Befehl mit dem Modul `setup` aus. Suchen Sie in der Ausgabe nach dem Abschnitt `ansible_local`. Zu diesem Zeitpunkt sollte dieser Abschnitt keine benutzerdefinierten Fakten enthalten.

```
[student@workstation data-facts]$ ansible webserver -m setup
servera.lab.example.com | SUCCESS => {
  "ansible_facts": {
    ...output omitted...
    "ansible_local": {}
  },
  "changed": false
}
```

- 6. Verifizieren Sie vor Ausführung des Playbooks, dass dessen Syntax korrekt ist, indem Sie `ansible-playbook --syntax-check` ausführen. Wenn Fehler gemeldet werden,

**Kapitel 3 |** Verwalten von Variablen und Fakten

korrigieren Sie sie, bevor Sie mit dem nächsten Schritt fortfahren. Es sollte eine Ausgabe ähnlich der folgenden angezeigt werden:

```
[student@workstation data-facts]$ ansible-playbook --syntax-check setup_facts.yml  
playbook: setup_facts.yml
```

- 7. Führen Sie das Playbook `setup_facts.yml` aus.

```
[student@workstation data-facts]$ ansible-playbook setup_facts.yml  
  
PLAY [Install remote facts] *****  
  
TASK [Gathering Facts] *****  
ok: [servera.lab.example.com]  
  
TASK [Create the remote directory] *****  
changed: [servera.lab.example.com]  
  
TASK [Install the new facts] *****  
changed: [servera.lab.example.com]  
  
PLAY RECAP *****  
servera.lab.example.com : ok=3    changed=2    unreachable=0    failed=0
```

- 8. Jetzt können Sie das Haupt-Playbook erstellen, in dem zum Konfigurieren von `servera` sowohl Standard- als auch benutzerdefinierte Fakten verwendet werden. In den nächsten Schritten fügen Sie der Playbook-Datei Einträge hinzu. Erstellen Sie das Playbook `playbook.yml` mit dem folgenden Inhalt:

```
---  
- name: Install Apache and starts the service  
  hosts: webserver
```

- 9. Setzen Sie die Bearbeitung der Datei `playbook.yml` fort, indem Sie die erste Aufgabe erstellen, die das Paket `httpd` installiert. Verwenden Sie für den Namen des Pakets das benutzerdefinierte Fakt.

```
tasks:  
  - name: Install the required package  
    yum:  
      name: "{{ ansible_facts['ansible_local']['custom']['general']['package'] }}"  
      state: latest
```

- 10. Erstellen Sie eine weitere Aufgabe, die mit dem benutzerdefinierten Fakt den httpd-Service startet.

```
- name: Start the service
  service:
    name: "{{ ansible_facts['ansible_local']['custom']['general']['service'] }}"
    state: "{{ ansible_facts['ansible_local']['custom']['general']['state'] }}"
    enabled: "{{ ansible_facts['ansible_local']['custom']['general']['enabled'] }}
```

- 11. Wenn alle Aufgaben erstellt sind, sollte das vollständige Playbook wie folgt aussehen. Überprüfen Sie das Playbook und stellen Sie sicher, dass alle Aufgaben definiert sind.

```
---
- name: Install Apache and starts the service
  hosts: webserver

  tasks:
    - name: Install the required package
      yum:
        name: "{{ ansible_facts['ansible_local']['custom']['general']['package'] }}"
        state: latest

    - name: Start the service
      service:
        name: "{{ ansible_facts['ansible_local']['custom']['general']['service'] }}"
        state: "{{ ansible_facts['ansible_local']['custom']['general']['state'] }}"
        enabled: "{{ ansible_facts['ansible_local']['custom']['general']['enabled'] }}
```

- 12. Überprüfen Sie vor der Ausführung des Playbooks mit einem Ad-hoc-Befehl, dass der httpd-Service aktuell nicht auf servera ausgeführt wird.

```
[student@workstation data-facts]$ ansible servera.lab.example.com -m command \
> -a 'systemctl status httpd'
servera.lab.example.com | FAILED | rc=4 >>
Unit httpd.service could not be found.non-zero return code
```

- 13. Verifizieren Sie die Syntax des Playbooks, indem Sie `ansible-playbook --syntax-check` ausführen. Wenn Fehler gemeldet werden, korrigieren Sie sie, bevor Sie mit dem nächsten Schritt fortfahren. Es sollte eine Ausgabe ähnlich der folgenden angezeigt werden:

```
[student@workstation data-facts]$ ansible-playbook --syntax-check playbook.yml
playbook: playbook.yml
```

**Kapitel 3 |** Verwalten von Variablen und Fakten

- 14. Führen Sie das Playbook mit dem Befehl `ansible-playbook` aus. Achten Sie auf die Ausgabe, während Ansible das Paket installiert und anschließend den Service aktiviert.

```
[student@workstation data-facts]$ ansible-playbook playbook.yml

PLAY [Install Apache and start the service] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Install the required package] ****
changed: [servera.lab.example.com]

TASK [Start the service] ****
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com      : ok=3    changed=2    unreachable=0    failed=0
```

- 15. Führen Sie einen Ad-hoc-Befehl aus, um `systemctl` auszuführen, um zu bestimmen, ob der Service `httpd` nun auf `servera` ausgeführt wird.

```
[student@workstation data-facts]$ ansible servera.lab.example.com -m command \
> -a 'systemctl status httpd'
servera.lab.example.com | CHANGED | rc=0 >>
● httpd.service - The Apache HTTP Server
  Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset:
  disabled)
  Active: active (running) since Mon 2019-05-27 07:50:55 EDT; 50s ago
    Docs: man:httpd.service(8)
  Main PID: 11603 (httpd)
    Status: "Running, listening on: port 80"
       Tasks: 213 (limit: 4956)
      Memory: 24.1M
     CGroup: /system.slice/httpd.service
...output omitted...
```

## Beenden

Führen Sie auf `workstation` das Skript `lab data-facts finish` aus, um diese Übung zu bereinigen.

```
[student@workstation ~]$ lab data-facts finish
```

Hiermit ist die angeleitete Übung beendet.

## ► Praktische Übung

# Verwalten von Variablen und Fakten

### Performance-Checkliste

In diesem Lab schreiben und führen sie ein Ansible Playbook aus, das Variablen, Geheimnisse und Fakten verwendet.

### Ergebnisse

Sie sollten in der Lage sein, Variablen zu definieren und Fakten in einem Playbook zu verwenden sowie die in einer verschlüsselten Datei definierten Variablen zu verwenden.

### Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab data-review start` aus. Das Skript erstellt das Arbeitsverzeichnis `/home/student/data-review` und füllt es mit einer Ansible-Konfigurationsdatei und dem Hostinventar. Der verwaltete Host `serverb.lab.example.com` ist in diesem Inventar als ein Mitglied der Hostgruppe `webserver` definiert. Ein Entwickler hat Sie gebeten, ein Ansible Playbook zu schreiben, um die Einrichtung einer Webserverumgebung auf `serverb.lab.example.com` zu automatisieren, der den Benutzerzugriff auf die zugehörige Website mittels grundlegender Authentifizierung steuert.

Das Unterverzeichnis `files` enthält:

- die Konfigurationsdatei `httpd.conf` für den Apache-Webservice für die Basisauthentifizierung
- die Datei `.htaccess`, die verwendet werden kann, um den Zugriff auf das Dokumentenstammverzeichnis des Webservers zu steuern
- die Datei `htpasswd` mit Anmelddaten für zulässige Benutzer

```
[student@workstation ~]$ lab data-review start
```

### Anweisungen

1. Erstellen Sie im Arbeitsverzeichnis das Playbook `playbook.yml`, und fügen Sie die Hostgruppe `webserver` als verwalteten Host hinzu. Definieren Sie die folgenden Play-Variablen:

## Variablen

| Variable       | Werte                        |
|----------------|------------------------------|
| firewall_pkg   | firewalld                    |
| firewall_svc   | firewalld                    |
| web_pkg        | httpd                        |
| web_svc        | httpd                        |
| ssl_pkg        | mod_ssl                      |
| httpdconf_src  | files/httpd.conf             |
| httpdconf_dest | /etc/httpd/conf/httpd.conf   |
| htaccess_src   | files/.htaccess              |
| secrets_dir    | /etc/httpd/secrets           |
| secrets_src    | files/htpasswd               |
| secrets_dest   | "{{ secrets_dir }}/htpasswd" |
| web_root       | /var/www/html                |

2. Fügen Sie dem Play den Abschnitt tasks hinzu. Schreiben Sie eine Aufgabe, durch die sichergestellt wird, dass die neueste Version der erforderlichen Pakete installiert ist. Diese Pakete werden durch die Variablen `firewall_pkg`, `web_pkg`, und `ssl_pkg` definiert.
3. Fügen Sie dem Playbook eine zweite Aufgabe hinzu, die sicherstellt, dass die durch die Variable `httpdconf_src` angegebene Datei (mit dem Modul `copy`) an den durch die Variable `httpdconf_dest` angegebenen Speicherort auf dem verwalteten Host kopiert wurde. Die Datei sollte dem Benutzer `root` und der Gruppe `root` gehören. Legen Sie auch 0644 als Dateiberechtigungen fest.
4. Fügen Sie eine dritte Aufgabe hinzu, die das Modul `file` verwendet, um das von der Variablen `secrets_dir` angegebene Verzeichnis auf dem verwalteten Host zu erstellen. Dieses Verzeichnis enthält die Passworddateien, die für die grundlegende Authentifizierung der Webservices verwendet werden. Die Datei sollte dem Benutzer `apache` und der Gruppe `apache` gehören. Set 0500 as the file permissions.
5. Fügen Sie eine vierte Aufgabe hinzu, die das Modul `copy` verwendet, um eine `htpasswd`-Datei zu platzieren, die für die grundlegende Authentifizierung von Webbenutzern verwendet wird. Die Quelle sollte durch die Variable `secrets_src` definiert werden. Das Ziel sollte durch die Variable `secrets_dest` definiert werden. Die Datei sollte dem Benutzer und der Gruppe `apache` gehören. Legen Sie 0400 als Dateiberechtigungen fest.
6. Fügen Sie eine fünfte Aufgabe hinzu, die das Modul `copy` verwendet, um die Datei `.htaccess` im Dokumentenstammverzeichnis des Webservers zu erstellen. Kopieren Sie die durch die Variable `htaccess_src` angegebene Datei nach `{{ web_root }}/.htaccess`. Die Datei sollte dem Benutzer `apache` und der Gruppe `apache` gehören. Legen Sie 0400 als Dateiberechtigungen fest.

7. Fügen Sie eine sechste Aufgabe hinzu, die das Modul `copy` verwendet, um die Webinhaltsdatei `index.html` im Verzeichnis zu erstellen, das von der Variablen `web_root` angegeben wurde. Die Datei sollte die Meldung „`HOSTNAME (IPADDRESS) has been customized by Ansible`“ enthalten, wobei `HOSTNAME` der vollqualifizierte Hostname des verwalteten Hosts und `IPADDRESS` die zugehörige IPv4-IP-Adresse ist. Verwenden Sie die Option `content` für das Modul `copy`, um den Inhalt der Datei anzugeben, und die Ansible-Fakten, um den Hostnamen und die IP-Adresse anzugeben.
8. Fügen Sie eine siebte Aufgabe hinzu, die das Modul `service` verwendet, um den Firewall-Service auf dem verwalteten Host zu aktivieren und zu starten.
9. Fügen Sie eine achte Aufgabe hinzu, die das Modul `firewalld` verwendet, um den `https`-Service zuzulassen, damit auf Webservices auf dem verwalteten Host zugreifen können. Diese Firewall-Änderung sollte dauerhaft sein und sofort erfolgen.
10. Fügen Sie eine finale Aufgabe hinzu, die das Modul `service` verwendet, um den Webservice auf dem verwalteten Host zu aktivieren und zu starten, damit alle Konfigurationsänderungen wirksam werden. Der Name des Webservices wird durch die Variable `web_svc` definiert.
11. Definieren Sie ein zweites Play für `localhost`, mit dem die Authentifizierung beim Webserver getestet wird. Es benötigt keine Rechteerweiterung. Definieren Sie eine Variable mit dem Namen `web_user` mit dem Wert `guest`.
12. Fügen Sie dem Play eine Anweisung hinzu, die zusätzliche Variablen aus einer Variablenliste mit dem Namen `vars/secret.yml` hinzufügt. Diese Datei enthält eine Variable namens `web_pass`, die das Passwort für den Webbenutzer angibt. Sie werden diese Datei später im Lab erstellen.  
Legen Sie den Beginn der Aufgabenliste fest.
13. Fügen Sie dem zweiten Play zwei Aufgaben hinzu.  
Die erste verwendet das Modul `uri` zum Anfordern von Inhalten aus `https://serverb.lab.example.com` unter Verwendung der Basisauthentifizierung. Verwenden Sie die Variablen `web_user` und `web_pass`, um sich beim Webserver zu authentifizieren. Beachten Sie, dass dem Zertifikat von `serverb` nicht vertraut wird. Daher müssen Sie die Validierung des Zertifikats vermeiden. Die Aufgabe sollte verifizieren, dass der HTTP-Statuscode `200` zurückgegeben wird. Konfigurieren Sie die Aufgabe so, dass der zurückgegebene Inhalt in die Aufgabenergebnis-Variable eingefügt wird. Registrieren Sie das Aufgabenergebnis in einer Variablen.  
Die zweite Aufgabe verwendet das Modul `debug` zum Drucken des vom Webserver zurückgegebenen Inhalts.
14. Erstellen Sie eine mit Ansible Vault verschlüsselte Datei mit dem Namen `vars/secret.yml`. Verwenden Sie das Passwort `redhat`, um sie zu verschlüsseln. Die Variable `web_pass` sollte auf `redhat` festgelegt werden, was dem Passwort des Webbenutzers entspricht.
15. Führen Sie das Playbook `playbook.yml` aus. Stellen Sie sicher, dass der Inhalt erfolgreich vom Webserver zurückgegeben wurde und mit dem übereinstimmt, was in einer früheren Aufgabe konfiguriert wurde.

## Bewertung

Führen Sie auf `workstation` den Befehl `lab data-review grade` aus, um den Erfolg dieser Übung zu bestätigen. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie das Skript so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab data-review grade
```

## Beenden

Führen Sie auf `workstation` den Befehl `lab data-review finish` aus, um diese Übung zu bereinigen.

```
[student@workstation ~]$ lab data-review finish
```

Hiermit ist die praktische Übung abgeschlossen.

## ► Lösung

# Verwalten von Variablen und Fakten

### Performance-Checkliste

In diesem Lab schreiben und führen sie ein Ansible Playbook aus, das Variablen, Geheimnisse und Fakten verwendet.

### Ergebnisse

Sie sollten in der Lage sein, Variablen zu definieren und Fakten in einem Playbook zu verwenden sowie die in einer verschlüsselten Datei definierten Variablen zu verwenden.

### Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab data-review start` aus. Das Skript erstellt das Arbeitsverzeichnis `/home/student/data-review` und füllt es mit einer Ansible-Konfigurationsdatei und dem Hostinventar. Der verwaltete Host `serverb.lab.example.com` ist in diesem Inventar als ein Mitglied der Hostgruppe `webserver` definiert. Ein Entwickler hat Sie gebeten, ein Ansible Playbook zu schreiben, um die Einrichtung einer Webserverumgebung auf `serverb.lab.example.com` zu automatisieren, der den Benutzerzugriff auf die zugehörige Website mittels grundlegender Authentifizierung steuert.

Das Unterverzeichnis `files` enthält:

- die Konfigurationsdatei `httpd.conf` für den Apache-Webservice für die Basisauthentifizierung
- die Datei `.htaccess`, die verwendet werden kann, um den Zugriff auf das Dokumentenstammverzeichnis des Webservers zu steuern
- die Datei `htpasswd` mit Anmelddaten für zulässige Benutzer

```
[student@workstation ~]$ lab data-review start
```

### Anweisungen

1. Erstellen Sie im Arbeitsverzeichnis das Playbook `playbook.yml`, und fügen Sie die Hostgruppe `webserver` als verwalteten Host hinzu. Definieren Sie die folgenden Play-Variablen:

## Variablen

| Variable       | Werte                        |
|----------------|------------------------------|
| firewall_pkg   | firewalld                    |
| firewall_svc   | firewalld                    |
| web_pkg        | httpd                        |
| web_svc        | httpd                        |
| ssl_pkg        | mod_ssl                      |
| httpdconf_src  | files/httpd.conf             |
| httpdconf_dest | /etc/httpd/conf/httpd.conf   |
| htaccess_src   | files/.htaccess              |
| secrets_dir    | /etc/httpd/secrets           |
| secrets_src    | files/htpasswd               |
| secrets_dest   | "{{ secrets_dir }}/htpasswd" |
| web_root       | /var/www/html                |

- 1.1. Wechseln Sie in das Arbeitsverzeichnis/home/student/data-review.

```
[student@workstation ~]$ cd ~/data-review
[student@workstation data-review]$
```

- 1.2. Erstellen Sie die Playbook-Datei playbook.yml, und bearbeiten Sie sie in einem Texteditor. Der Anfang der Datei sollte folgendermaßen aussehen:

```
---
- name: install and configure webserver with basic auth
  hosts: webserver
  vars:
    firewall_pkg: firewalld
    firewall_svc: firewalld
    web_pkg: httpd
    web_svc: httpd
    ssl_pkg: mod_ssl
    httpdconf_src: files/httpd.conf
    httpdconf_dest: /etc/httpd/conf/httpd.conf
    htaccess_src: files/.htaccess
    secrets_dir: /etc/httpd/secrets
    secrets_src: files/htpasswd
    secrets_dest: "{{ secrets_dir }}/htpasswd"
    web_root: /var/www/html
```

2. Fügen Sie dem Play den Abschnitt tasks hinzu. Schreiben Sie eine Aufgabe, durch die sichergestellt wird, dass die neueste Version der erforderlichen Pakete installiert ist. Diese Pakete werden durch die Variablen `firewall_pkg`, `web_pkg`, und `ssl_pkg` definiert.

- 2.1. Definieren Sie den Anfang des Abschnitts tasks, indem Sie dem Playbook die folgende Zeile hinzufügen:

```
tasks:
```

- 2.2. Fügen Sie dem Playbook die folgenden Zeilen hinzu, um eine Aufgabe zu definieren, die das Modul `yum` verwendet, um die erforderlichen Pakete zu installieren.

```
- name: latest version of necessary packages installed
  yum:
    name:
      - "{{ firewall_pkg }}"
      - "{{ web_pkg }}"
      - "{{ ssl_pkg }}"
    state: latest
```

3. Fügen Sie dem Playbook eine zweite Aufgabe hinzu, die sicherstellt, dass die durch die Variable `httpdconf_src` angegebene Datei (mit dem Modul `copy`) an den durch die Variable `httpdconf_dest` angegebenen Speicherort auf dem verwalteten Host kopiert wurde. Die Datei sollte dem Benutzer `root` und der Gruppe `root` gehören. Legen Sie auch 0644 als Dateiberechtigungen fest.

Fügen Sie dem Playbook die folgenden Zeilen hinzu, um eine Aufgabe zu definieren, die das Modul `copy` verwendet, um die Inhalte der von der Variablen `httpdconf_src` definierten Datei an den Speicherort zu kopieren, der von der Variablen `httpdconf_dest` angegeben wurde.

```
- name: configure web service
  copy:
    src: "{{ httpdconf_src }}"
    dest: "{{ httpdconf_dest }}"
    owner: root
    group: root
    mode: 0644
```

4. Fügen Sie eine dritte Aufgabe hinzu, die das Modul `file` verwendet, um das von der Variablen `secrets_dir` angegebene Verzeichnis auf dem verwalteten Host zu erstellen. Dieses Verzeichnis enthält die Passworddateien, die für die grundlegende Authentifizierung der Webservices verwendet werden. Die Datei sollte dem Benutzer `apache` und der Gruppe `apache` gehören. Set 0500 as the file permissions.

Fügen Sie dem Playbook die folgenden Zeilen hinzu, um eine Aufgabe zu definieren, die das Modul `file` verwendet, um das von der Variablen `secrets_dir` definierte Verzeichnis zu erstellen.

```
- name: secrets directory exists
  file:
    path: "{{ secrets_dir }}"
    state: directory
    owner: apache
    group: apache
    mode: 0500
```

5. Fügen Sie eine vierte Aufgabe hinzu, die das Modul `copy` verwendet, um eine `htpasswd`-Datei zu platzieren, die für die grundlegende Authentifizierung von Webbenutzern verwendet wird. Die Quelle sollte durch die Variable `secrets_src` definiert werden. Das Ziel sollte durch die Variable `secrets_dest` definiert werden. Die Datei sollte dem Benutzer und der Gruppe `apache` gehören. Legen Sie `0400` als Dateiberechtigungen fest.

```
- name: htpasswd file exists
  copy:
    src: "{{ secrets_src }}"
    dest: "{{ secrets_dest }}"
    owner: apache
    group: apache
    mode: 0400
```

6. Fügen Sie eine fünfte Aufgabe hinzu, die das Modul `copy` verwendet, um die Datei `.htaccess` im Dokumentenstammverzeichnis des Webservers zu erstellen. Kopieren Sie die durch die Variable `htaccess_src` angegebene Datei nach `{{ web_root }}/.htaccess`. Die Datei sollte dem Benutzer `apache` und der Gruppe `apache` gehören. Legen Sie `0400` als Dateiberechtigungen fest.

Fügen Sie dem Playbook die folgenden Zeilen hinzu, um eine Aufgabe zu definieren, die das Modul `copy` verwendet, um die Datei `.htaccess` mit der Datei zu erstellen, die von der Variablen `htaccess_src` definiert wurde.

```
- name: .htaccess file installed in docroot
  copy:
    src: "{{ htaccess_src }}"
    dest: "{{ web_root }}/.htaccess"
    owner: apache
    group: apache
    mode: 0400
```

7. Fügen Sie eine sechste Aufgabe hinzu, die das Modul `copy` verwendet, um die Webinhaltsdatei `index.html` im Verzeichnis zu erstellen, das von der Variablen `web_root` angegeben wurde. Die Datei sollte die Meldung „`HOSTNAME (IPADDRESS) has been customized by Ansible`“ enthalten, wobei `HOSTNAME` der vollqualifizierte Hostname des verwalteten Hosts und `IPADDRESS` die zugehörige IPv4-IP-Adresse ist. Verwenden Sie die Option `content` für das Modul `copy`, um den Inhalt der Datei anzugeben, und die Ansible-Fakten, um den Hostnamen und die IP-Adresse anzugeben.

Fügen Sie dem Playbook die folgenden Zeilen hinzu, um eine Aufgabe zu definieren, die das Modul `copy` verwendet, um die Datei `index.html` in dem Verzeichnis zu erstellen, das von der Variablen `web_root` definiert wurde. Füllen Sie die Datei mit dem angegebenen Inhalt

mittels der vom verwalteten Host abgerufenen Ansible-Fakten `ansible_facts['fqdn']` und `ansible_facts['default_ipv4']['address']`.

```
- name: create index.html
  copy:
    content: "{{ ansible_facts['fqdn'] }} {{ ansible_facts['default_ipv4'] [
      'address'] }} has been customized by Ansible.\n"
    dest: "{{ web_root }} /index.html"
```

8. Fügen Sie eine siebte Aufgabe hinzu, die das Modul `service` verwendet, um den Firewall-Service auf dem verwalteten Host zu aktivieren und zu starten.

Fügen Sie dem Playbook die folgenden Zeilen hinzu, um eine Aufgabe zu definieren, die das Modul `service` verwendet, um den Firewall-Service zu aktivieren und zu starten.

```
- name: firewall service enabled and started
  service:
    name: "{{ firewall_svc }}"
    state: started
    enabled: true
```

9. Fügen Sie eine achte Aufgabe hinzu, die das Modul `firewalld` verwendet, um den `https`-Service zuzulassen, damit auf Webservices auf dem verwalteten Host zugreifen können. Diese Firewall-Änderung sollte dauerhaft sein und sofort erfolgen.

Fügen Sie dem Playbook die folgenden Zeilen hinzu, um eine Aufgabe zu definieren, die das Modul `firewalld` verwendet, um den HTTPS-Port für den Webservice zu öffnen.

```
- name: open the port for the web server
  firewalld:
    service: https
    state: enabled
    immediate: true
    permanent: true
```

10. Fügen Sie eine finale Aufgabe hinzu, die das Modul `service` verwendet, um den Webservice auf dem verwalteten Host zu aktivieren und zu starten, damit alle Konfigurationsänderungen wirksam werden. Der Name des Webservices wird durch die Variable `web_svc` definiert.

```
- name: web service enabled and started
  service:
    name: "{{ web_svc }}"
    state: started
    enabled: true
```

11. Definieren Sie ein zweites Play für `localhost`, mit dem die Authentifizierung beim Webserver getestet wird. Es benötigt keine Rechteerweiterung. Definieren Sie eine Variable mit dem Namen `web_user` mit dem Wert `guest`.

- 11.1. Fügen Sie die folgende Zeile hinzu, um den Beginn eines zweiten Plays zu definieren. Beachten Sie, dass es keine Einrückung gibt.

```
- name: test web server with basic auth
```

**Kapitel 3 |** Verwalten von Variablen und Fakten

- 11.2. Fügen Sie die folgende Zeile hinzu, um anzugeben, dass sich das Play auf den verwalteten Host `localhost` bezieht.

```
hosts: localhost
```

- 11.3. Fügen Sie die folgende Zeile hinzu, um die Rechteerweiterung zu deaktivieren.

```
become: no
```

- 11.4. Fügen Sie die folgenden Zeilen hinzu, um eine Variablenliste und die Variable `web_user` zu definieren.

```
vars:  
  web_user: guest
```

- 12.** Fügen Sie dem Play eine Anweisung hinzu, die zusätzliche Variablen aus einer Variablendatei mit dem Namen `vars/secret.yml` hinzufügt. Diese Datei enthält eine Variable namens `web_pass`, die das Passwort für den Webbenutzer angibt. Sie werden diese Datei später im Lab erstellen.

Legen Sie den Beginn der Aufgabenliste fest.

- 12.1. Fügen Sie mit dem Keyword `vars_files` dem Playbook die folgenden Zeilen hinzu, um Ansible anzuweisen, die in der Variablendatei `vars/secret.yml` befindlichen Variablen zu verwenden.

```
vars_files:  
  - vars/secret.yml
```

- 12.2. Fügen Sie die folgende Zeile hinzu, um den Beginn der Liste `tasks` zu definieren.

```
tasks:
```

- 13.** Fügen Sie dem zweiten Play zwei Aufgaben hinzu.

Die erste verwendet das Modul `uri` zum Anfordern von Inhalten aus `https://serverb.lab.example.com` unter Verwendung der Basisauthentifizierung. Verwenden Sie die Variablen `web_user` und `web_pass`, um sich beim Webserver zu authentifizieren. Beachten Sie, dass dem Zertifikat von `serverb` nicht vertraut wird. Daher müssen Sie die Validierung des Zertifikats vermeiden. Die Aufgabe sollte verifizieren, dass der HTTP-Statuscode 200 zurückgegeben wird. Konfigurieren Sie die Aufgabe so, dass der zurückgegebene Inhalt in die Aufgabenergebnis-Variablen eingefügt wird. Registrieren Sie das Aufgabenergebnis in einer Variablen.

Die zweite Aufgabe verwendet das Modul `debug` zum Drucken des vom Webserver zurückgegebenen Inhalts.

- 13.1. Fügen Sie die folgenden Zeilen hinzu, um die Aufgabe zu erstellen, mit der der Webservice über den Kontrollknoten verifiziert werden. Rücken Sie die erste Zeile unbedingt mit vier Leerzeichen ein.

```
- name: connect to web server with basic auth  
  uri:  
    url: https://serverb.lab.example.com
```

```
validate_certs: no
force_basic_auth: yes
user: "{{ web_user }}"
password: "{{ web_pass }}"
return_content: yes
status_code: 200
register: auth_test
```

- 13.2. Erstellen Sie die zweite Aufgabe mithilfe des Moduls „debug“. Der vom Webserver zurückgegebene Inhalt wird der registrierten Variablen als Schlüssel **content** hinzugefügt.

```
- debug:
  var: auth_test.content
```

- 13.3. Wenn Sie fertig sind, sollte das Playbook wie folgt aussehen:

```
---
- name: install and configure webserver with basic auth
hosts: webserver
vars:
  firewall_pkg: firewalld
  firewall_svc: firewalld
  web_pkg: httpd
  web_svc: httpd
  ssl_pkg: mod_ssl
  httpdconf_src: files/httpd.conf
  httpdconf_dest: /etc/httpd/conf/httpd.conf
  htaccess_src: files/.htaccess
  secrets_dir: /etc/httpd/secrets
  secrets_src: files/htpasswd
  secrets_dest: "{{ secrets_dir }}/htpasswd"
  web_root: /var/www/html
tasks:
  - name: latest version of necessary packages installed
    yum:
      name:
        - "{{ firewall_pkg }}"
        - "{{ web_pkg }}"
        - "{{ ssl_pkg }}"
      state: latest

  - name: configure web service
    copy:
      src: "{{ httpdconf_src }}"
      dest: "{{ httpdconf_dest }}"
      owner: root
      group: root
      mode: 0644

  - name: secrets directory exists
    file:
      path: "{{ secrets_dir }}"
      state: directory
```

```
owner: apache
group: apache
mode: 0500

- name: htpasswd file exists
  copy:
    src: "{{ secrets_src }}"
    dest: "{{ secrets_dest }}"
    owner: apache
    group: apache
    mode: 0400

- name: .htaccess file installed in docroot
  copy:
    src: "{{ htaccess_src }}"
    dest: "{{ web_root }}/.htaccess"
    owner: apache
    group: apache
    mode: 0400

- name: create index.html
  copy:
    content: "{{ ansible_facts['fqdn'] }} ({{ ansible_facts['default_ipv4'] }}['address']) has been customized by Ansible.\n"
    dest: "{{ web_root }}/index.html"

- name: firewall service enable and started
  service:
    name: "{{ firewall_svc }}"
    state: started
    enabled: true

- name: open the port for the web server
  firewalld:
    service: https
    state: enabled
    immediate: true
    permanent: true

- name: web service enabled and started
  service:
    name: "{{ web_svc }}"
    state: started
    enabled: true

- name: test web server with basic auth
  hosts: localhost
  become: no
  vars:
    - web_user: guest
  vars_files:
    - vars/secret.yml
  tasks:
    - name: connect to web server with basic auth
      uri:
```

**Kapitel 3 |** Verwalten von Variablen und Fakten

```
url: https://serverb.lab.example.com
validate_certs: no
force_basic_auth: yes
user: "{{ web_user }}"
password: "{{ web_pass }}"
return_content: yes
status_code: 200
register: auth_test

- debug:
  var: auth_test.content
```

13.4. Speichern und schließen Sie die Datei `playbook.yml`.

14. Erstellen Sie eine mit Ansible Vault verschlüsselte Datei mit dem Namen `vars/secret.yml`. Verwenden Sie das Passwort `redhat`, um sie zu verschlüsseln. Die Variable `web_pass` sollte auf `redhat` festgelegt werden, was dem Passwort des Webbenutzers entspricht.

14.1. Erstellen Sie ein Unterverzeichnis mit dem Namen `vars` im Arbeitsverzeichnis.

```
[student@workstation data-review]$ mkdir vars
```

14.2. Erstellen Sie die verschlüsselte Variablendatei `vars/secret.yml` mit Ansible Vault. Legen Sie das Passwort für die verschlüsselte Datei auf `redhat` fest.

```
[student@workstation data-review]$ ansible-vault create vars/secret.yml
New Vault password: redhat
Confirm New Vault password: redhat
```

14.3. Fügen Sie der Datei die folgende Variablendefinition hinzu.

```
web_pass: redhat
```

14.4. Speichern und schließen Sie die Datei.

15. Führen Sie das Playbook `playbook.yml` aus. Stellen Sie sicher, dass der Inhalt erfolgreich vom Webserver zurückgegeben wurde und mit dem übereinstimmt, was in einer früheren Aufgabe konfiguriert wurde.

15.1. Verifizieren Sie vor Ausführung des Playbooks, dass dessen Syntax korrekt ist, indem Sie `ansible-playbook --syntax-check` ausführen. Verwenden Sie `--ask-vault-pass`, um aufgefordert zu werden, das Vault-Passwort einzugeben. Geben Sie `redhat` ein, wenn Sie aufgefordert werden, das Passwort einzugeben. Wenn Fehler gemeldet werden, korrigieren Sie sie, bevor Sie mit dem nächsten Schritt fortfahren. Es sollte eine Ausgabe ähnlich der folgenden angezeigt werden:

```
[student@workstation data-review]$ ansible-playbook --syntax-check \
> --ask-vault-pass playbook.yml
Vault password: redhat

playbook: playbook.yml
```

### Kapitel 3 | Verwalten von Variablen und Fakten

- 15.2. Führen Sie mit dem Befehl `ansible-playbook` das Playbook mit der Option `--ask-vault-pass` aus. Geben Sie `redhat` ein, wenn Sie aufgefordert werden, das Passwort einzugeben.

```
[student@workstation data-review]$ ansible-playbook playbook.yml --ask-vault-pass
Vault password: redhat
PLAY [Install and configure webserver with basic auth] ****
...output omitted...

TASK [connect to web server with basic auth] ****
ok: [localhost]

TASK [debug] ****
ok: [localhost] => {
    "auth_test.content": "serverb.lab.example.com (172.25.250.11) has been
    customized by Ansible.\n"
}

PLAY RECAP ****
localhost                  : ok=3      changed=0      unreachable=0      failed=0
serverb.lab.example.com   : ok=10     changed=8      unreachable=0      failed=0
```

## Bewertung

Führen Sie auf `workstation` den Befehl `lab data-review grade` aus, um den Erfolg dieser Übung zu bestätigen. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie das Skript so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab data-review grade
```

## Beenden

Führen Sie auf `workstation` den Befehl `lab data-review finish` aus, um diese Übung zu bereinigen.

```
[student@workstation ~]$ lab data-review finish
```

Hiermit ist die praktische Übung abgeschlossen.

# Zusammenfassung

---

In diesem Kapitel wurden die folgenden Themen behandelt:

- Ansible-*Variablen* ermöglichen Administratoren die Wiederverwendung von Werten in Dateien eines Ansible-Projekts.
- Variablen können für Hosts und Hostgruppen in der Inventardatei definiert werden.
- Variablen können für Playbooks mithilfe von Fakten und externen Dateien definiert werden. Sie können auch in der Befehlszeile definiert werden.
- Mit dem Keyword `register` kann die Ausgabe eines Befehls in einer Variable erfasst werden.
- Ansible Vault ist eine Möglichkeit, um sensible Dateien, wie Passwort-Hash-Werte und private Schlüssel, für Bereitstellungen mit Ansible-Playbooks zu verwenden.
- Ansible-*Fakten* sind Variablen, die von Ansible automatisch auf einem verwalteten Host ermittelt werden.



## Kapitel 4

# Implementieren der Aufgabensteuerung

### Ziel

Aufgabensteuerung, Handler und Aufgabenfehler in Ansible-Playbooks verwalten.

### Ziele

- Verwenden von Loops zum Schreiben effizienter Aufgaben und Verwenden von Bedingungen, um zu steuern, wann Aufgaben ausgeführt werden sollen.
- Implementieren von Aufgaben, die nur ausgeführt wird, wenn eine andere Aufgabe den verwalteten Host ändert.
- Festlegen, was passiert, wenn eine Aufgabe fehlschlägt und unter welchen Bedingungen eine Aufgabe fehlschlägt.

### Abschnitte

- Schreiben von Loops und bedingten Aufgaben (und angeleitete Übung)
- Implementieren von Handlern (und angeleitete Übung)
- Task-Fehler behandeln (und angeleitete Übung)

### Praktische Übung

- Implementieren der Aufgabensteuerung

# Schreiben von Loops und bedingten Aufgaben

## Ziele

Am Ende dieses Abschnitts sollten Sie in der Lage sein, Loops zu verwenden, um effiziente Aufgaben zu schreiben, und Bedingungen einzusetzen, um zu steuern, wann Aufgaben ausgeführt werden sollen.

## Aufgabeniteration mit Loops

Die Verwendung von Loops erspart es Administratoren, mehrere Aufgaben erstellen zu müssen, die dasselbe Modul verwenden. Statt z. B. fünf Aufgaben zu erstellen, um zu gewährleisten, dass fünf Benutzer vorhanden sind, können Sie eine Aufgabe erstellen, die über eine Liste von fünf Benutzern wiederholt wird, um sicherzustellen, dass alle vorhanden sind.

Ansible unterstützt die Iteration einer Aufgabe über eine Reihe von Elementen mithilfe des Keywords `loop`. Um eine Aufgabe zu wiederholen, können Sie Loops konfigurieren, um die einzelnen Elemente in einer Liste, die Inhalte aller Dateien in einer Liste, eine generierte Nummernfolge oder kompliziertere Strukturen zu verwenden. In diesem Abschnitt werden einfache Loops beschrieben, die eine Liste von Elementen durchlaufen. In der Dokumentation finden Sie weiterführende Loop-Szenarien.

## Einfache Loops

Mit einem einfachen Loop wird eine Aufgabe über eine Liste von Elementen wiederholt. Das Keyword `loop` wird zur Aufgabe hinzugefügt und übernimmt als Wert die Liste der Elemente, über welche die Aufgabe wiederholt werden soll. Die Loop-Variable `item` enthält den Wert, der für jede Iteration verwendet wird.

Betrachten Sie den folgenden Auszug, in dem zwei Mal das Modul `service` verwendet wird, um sicherzustellen, dass zwei Netzwerkservices ausgeführt werden:

```
- name: Postfix is running
  service:
    name: postfix
    state: started

- name: Dovecot is running
  service:
    name: dovecot
    state: started
```

Diese beiden Aufgaben können so umgeschrieben werden, dass sie einen einfachen Loop verwenden, damit nur eine Aufgabe benötigt wird, um sicherzustellen, dass beide Services ausgeführt werden:

```
- name: Postfix and Dovecot are running
  service:
    name: "{{ item }}"
    state: started
  loop:
    - postfix
    - dovecot
```

Die in dem Loop `loop` verwendete Liste kann von einer Variablen bereitgestellt werden. Im folgenden Beispiel enthält die Variable `mail_services` die Liste der Services, die ausgeführt werden müssen.

```
vars:
  mail_services:
    - postfix
    - dovecot

tasks:
  - name: Postfix and Dovecot are running
    service:
      name: "{{ item }}"
      state: started
    loop: "{{ mail_services }}"
```

## Durchläuft eine Liste von Hashes oder Dictionaries

Bei der Liste `loop` muss es sich nicht um eine Liste einfacher Werte handeln. Im folgenden Beispiel ist jedes Element in der Liste eigentlich ein Hash oder Dictionary. Jedes Hash oder Wörterbuch im Beispiel enthält die beiden Schlüssel `name` und `groups`, und der Wert der einzelnen Schlüssel kann in der aktuellen Schleifenvariable `item` mit den Variablen `item.name` bzw. `item.groups` abgerufen werden.

```
- name: Users exist and are in the correct groups
  user:
    name: "{{ item.name }}"
    state: present
    groups: "{{ item.groups }}"
  loop:
    - name: jane
      groups: wheel
    - name: joe
      groups: root
```

Das Ergebnis der vorherigen Aufgabe besteht darin, dass der Benutzer `jane` vorhanden und ein Mitglied der Gruppe `wheel` ist und dass der Benutzer `joe` vorhanden und ein Mitglied der Gruppe `root` ist.

## Vorherige Loop-Keywords

Vor Ansible 2.5 verwendeten die meisten Playbooks eine andere Syntax für Loops. Es wurden mehrere Loop-Keywords bereitgestellt, denen ein `with_`-Präfix vorangestellt war, gefolgt von dem Namen eines Ansible-Plugins zum Nachschlagen (ein erweitertes Feature, das in diesem Kurs

**Kapitel 4 |** Implementieren der Aufgabensteuerung

nicht ausführlich behandelt wird). Diese Syntax für das Looping kommt in vorhandenen Playbooks häufig vor, wird aber wahrscheinlich irgendwann in der Zukunft als abgekündigt gelten.

Einige Beispiele sind in der folgenden Tabelle aufgeführt:

**Früher verwendete Syntax von Ansible-Loops**

| Loop-Keyword  | Beschreibung                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| with_items    | Verhält sich wie das <code>loop</code> -Keyword für einfache Listen, z. B. eine Liste von Zeichenfolgen oder eine Liste von Hashes/Dictionaries. Anders als bei einer <code>loop</code> , wenn Listen von Listen an <code>with_items</code> bereitgestellt werden, werden sie zu einer einstufigen Liste zusammengefasst. Die Loop-Variable <code>item</code> enthält den Listeneintrag, der für jede Iteration verwendet wird. |
| with_file     | Dieses Keyword erfordert eine Liste von Dateinamen der Kontrollknoten. Die Loop-Variable <code>item</code> enthält während jeder Iteration den Inhalt einer entsprechenden Datei aus der Dateiliste.                                                                                                                                                                                                                            |
| with_sequence | Anstelle einer Liste erfordert dieses Keyword Parameter, um eine Liste von Werten basierend auf einer numerischen Sequenz zu generieren. Die Loop-Variable <code>item</code> enthält den Wert eines der generierten Elemente in der generierten Sequenz während jeder Iteration.                                                                                                                                                |

Ein Beispiel für `with_items` in einem Playbook wird unten gezeigt:

```
vars:
  data:
    - user0
    - user1
    - user2
tasks:
  - name: "with_items"
    debug:
      msg: "{{ item }}"
    with_items: "{{ data }}"
```

**Wichtig**

Seit Ansible 2.5 ist die empfohlene Methode zum Schreiben von Loops das Keyword `loop`.

Sie sollten jedoch vor allem die alte Syntax verstehen, insbesondere `with_items`, weil diese in bestehenden Playbooks weit verbreitet ist. Sie werden wahrscheinlich auf Playbooks und Rollen stoßen, die weiterhin `with_*` Keywords für das Looping verwenden.

Jede Aufgabe, in der die alte Syntax verwendet wird, kann konvertiert werden, um `loop` in Verbindung mit Ansible-Filters zu verwenden. Sie müssen dazu nicht wissen, wie Ansible-Filter verwendet werden. Unter Abschnitt Migrieren von `with_X` zu `Loop` [[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_loops.html#migrating-from-with-x-to-loop](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_loops.html#migrating-from-with-x-to-loop)] im *Ansible User*

Guide finden Sie eine gute Anleitung zum Konvertieren der alten Loops in die neue Syntax sowie Beispiele für das Erstellen von Loops für Elemente, die keine einfachen Listen sind.

Sie werden wahrscheinlich auf Aufgaben aus älteren Playbooks stoßen, die `with_*`-Keywords enthalten.

Fortgeschrittene Loop-Verfahren sind nicht Bestandteil dieses Kurses. Alle in diesem Kurs vorgesehenen Aufgaben für die Iteration können entweder mit den `with_items` oder den `loop` Keywords implementiert werden.

## Verwenden von Registervariablen mit Loops

Das Keyword `register` kann auch die Ausgabe einer Aufgabe erfassen, die sich in einem Loop befindet. Der folgende Ausschnitt zeigt die Struktur der Registervariablen aus einer Aufgabe, die sich in einem Loop befindet:

```
[student@workstation loopdemo]$ cat loop_register.yml
---
- name: Loop Register Test
  gather_facts: no
  hosts: localhost
  tasks:
    - name: Looping Echo Task
      shell: "echo This is my item: {{ item }}"
      loop:
        - one
        - two
      register: echo_results 1
    - name: Show echo_results variable
      debug:
        var: echo_results 2
```

- 1** Die Variable `echo_results` ist registriert.
- 2** Die Inhalte der `echo_results`-Variable werden auf dem Bildschirm angezeigt.

Wenn Sie das obige Playbook ausführen, erhalten Sie folgende Ausgabe:

```
[student@workstation loopdemo]$ ansible-playbook loop_register.yml
PLAY [Loop Register Test] ****
TASK [Looping Echo Task] ****
...output omitted...
TASK [Show echo_results variable] ****
ok: [localhost] => {
  "echo_results": { 1
    "changed": true,
    "msg": "All items completed",
    "results": [ 2
      { 3
        "_ansible_ignore_errors": null,
        ...output omitted...
        "changed": true,
        "cmd": "echo This is my item: one",
```

```
        "delta": "0:00:00.011865",
        "end": "2018-11-01 16:32:56.080433",
        "failed": false,
        ...output omitted...
        "item": "one",
        "rc": 0,
        "start": "2018-11-01 16:32:56.068568",
        "stderr": "",
        "stderr_lines": [],
        "stdout": "This is my item: one",
        "stdout_lines": [
            "This is my item: one"
        ]
    },
    {❶
        "_ansible_ignore_errors": null,
        ...output omitted...
        "changed": true,
        "cmd": "echo This is my item: two",
        "delta": "0:00:00.011142",
        "end": "2018-11-01 16:32:56.828196",
        "failed": false,
        ...output omitted...
        "item": "two",
        "rc": 0,
        "start": "2018-11-01 16:32:56.817054",
        "stderr": "",
        "stderr_lines": [],
        "stdout": "This is my item: two",
        "stdout_lines": [
            "This is my item: two"
        ]
    }
]
}
...output omitted...
```

- ❶ Das Zeichen { bedeutet, dass der Anfang der Variable echo\_results aus Schlüsselpaaren besteht.
- ❷ Der Schlüssel results enthält die Ergebnisse der vorherigen Aufgabe. Das [ Zeichen gibt den Beginn einer Liste an.
- ❸ Der Beginn der Aufgabenmetadaten für das erste Element (gekennzeichnet durch den Schlüssel item). Die Ausgabe des echo-Befehls befindet sich im Schlüssel stdout.
- ❹ Der Beginn der Aufgabenergebnis-Metadaten für das zweite Element.
- ❺ Das Zeichen ] zeigt das Ende der Liste results an.

Der Schlüssel results oben enthält eine Liste. Im Folgenden wird das Playbook so modifiziert, dass die zweite Aufgabe über diese Liste wiederholt wird:

```
[student@workstation loopdemo]$ cat new_loop_register.yml
---
- name: Loop Register Test
  gather_facts: no
  hosts: localhost
  tasks:
    - name: Looping Echo Task
      shell: "echo This is my item: {{ item }}"
      loop:
        - one
        - two
      register: echo_results

    - name: Show stdout from the previous task.
      debug:
        msg: "STDOUT from previous task: {{ item.stdout }}"
        loop: "{{ echo_results['results'] }}"
```

Nach der Ausführung des obigen Playbooks lautet die Ausgabe:

```
PLAY [Loop Register Test] ****
TASK [Looping Echo Task] ****
...output omitted...

TASK [Show stdout from the previous task.] ****
ok: [localhost] => (item={...output omitted...}) => {
    "msg": "STDOUT from previous task: This is my item: one"
}
ok: [localhost] => (item={...output omitted...}) => {
    "msg": "STDOUT from previous task: This is my item: two"
}
...output omitted...
```

## Bedingtes Ausführen von Aufgaben

Ansible kann *conditionals* verwenden, um Aufgaben oder Plays auszuführen, sofern bestimmte Bedingungen erfüllt sind. Beispielsweise kann eine Bedingung verwendet werden, um den verfügbaren Speicher auf einem verwalteten Host zu bestimmen, bevor Ansible einen Service installiert oder konfiguriert.

Bedingungen ermöglichen es Administratoren zwischen verwalteten Hosts zu differenzieren und ihnen funktionale Rollen aufgrund der erfüllten Bedingungen zuzuweisen. Playbook-Variablen, registrierte Variablen und Ansible-Fakten können mit Bedingungen geprüft werden. Es stehen Operatoren zum Vergleichen von Strings, numerischen Daten und Booleschen Werten zur Verfügung.

Die folgenden Szenarien veranschaulichen die Verwendung von bedingten Elementen in Ansible:

- Ein harter Grenzwert kann in einer Variablen (z. B. `min_memory`) definiert und mit dem verfügbaren Speicher auf einem verwalteten Host verglichen werden.

**Kapitel 4 |** Implementieren der Aufgabensteuerung

- Die Ausgabe des Befehls kann aufgenommen und von Ansible ausgewertet werden, um zu ermitteln, ob eine Aufgabe abgeschlossen wurde, bevor weitere Schritte ergriffen werden. Wenn beispielsweise ein Programm fehlerhaft ist, dann wird ein Batch übersprungen.
- Sie können Ansible-Fakten verwenden, um die Netzwerkkonfiguration von verwalteten Hosts zu bestimmen und um zu entscheiden, welche Vorlagendatei gesendet werden soll (zum Beispiel Netzwerkbündelung oder -Trunkierung).
- Die Anzahl der CPUs kann bewertet werden, um die ordnungsgemäße Einstellung eines Webservers festzulegen.
- Vergleichen Sie eine registrierte Variable mit einer vordefinierten Variablen, um festzustellen, ob sich ein Service geändert hat. Testen Sie beispielsweise die MD5-Prüfsumme einer Service-Konfigurationsdatei, um festzustellen, ob der Service geändert wurde.

## Aufgaben-Syntax mit Bedingungen

Mithilfe der `when`-Anweisung kann eine Aufgabe bedingt ausgeführt werden. Sie enthält als Wert die zu testende Bedingung. Wenn die Bedingung erfüllt ist, wird die Aufgabe ausgeführt. Wenn die Bedingung nicht erfüllt ist, wird die Aufgabe übersprungen.

Eine der einfachsten Bedingungen, die getestet werden kann, besteht darin, ob eine Boolesche Variable den Wert „true“ oder „false“ enthält. Die `when`-Anweisung im folgenden Beispiel bewirkt, dass die Aufgabe nur dann ausgeführt wird, wenn `run_my_task` den Wert „true“ enthält:

```
---
- name: Simple Boolean Task Demo
  hosts: all
  vars:
    run_my_task: true

  tasks:
    - name: httpd package is installed
      yum:
        name: httpd
      when: run_my_task
```

Das nächste Beispiel ist ein bisschen komplexer; es testet, ob die Variable `my_service` einen Wert enthält. Ist dies der Fall, wird der Wert von `my_service` als Name des zu installierenden Pakets verwendet. Wenn die Variable `my_service` nicht definiert ist, wird die Aufgabe ohne Fehler übersprungen.

```
---
- name: Test Variable is Defined Demo
  hosts: all
  vars:
    my_service: httpd

  tasks:
    - name: "{{ my_service }} package is installed"
      yum:
        name: "{{ my_service }}"
      when: my_service is defined
```

Die folgende Tabelle zeigt einige der Funktionen, die Administratoren bei der Arbeit mit Bedingungen verwenden können:

### Beispielbedingungen

| Funktion                                                                                   | Beispiel                                               |
|--------------------------------------------------------------------------------------------|--------------------------------------------------------|
| gleich (Wert ist eine Zeichenfolge)                                                        | <code>ansible_machine == "x86_64"</code>               |
| gleich (Wert ist numerisch)                                                                | <code>max_memory == 512</code>                         |
| kleiner als                                                                                | <code>min_memory &lt; 128</code>                       |
| größer als                                                                                 | <code>min_memory &gt; 256</code>                       |
| kleiner gleich                                                                             | <code>min_memory &lt;= 256</code>                      |
| größer gleich                                                                              | <code>min_memory &gt;= 512</code>                      |
| ungleich                                                                                   | <code>min_memory != 512</code>                         |
| Variable existiert                                                                         | <code>min_memory ist definiert</code>                  |
| Variable existiert nicht                                                                   | <code>min_memory ist nicht definiert</code>            |
| Boolesche Variable ist true. Die Werte von 1, True oder yes werden als true ausgewertet.   | <code>memory_available</code>                          |
| Boolesche Variable ist false. Die Werte von 0, False oder no werden als false ausgewertet. | <code>not memory_available</code>                      |
| Wert der ersten Variablen ist als Wert in Liste der zweiten Variablen enthalten            | <code>ansible_distribution in supported_distros</code> |

Der letzte Eintrag in der vorherigen Tabelle ist zunächst vielleicht verwirrend. Das folgende Beispiel zeigt, wie es funktioniert.

Im Beispiel wird die Variable `ansible_distribution`, die ein Fakt ist, während der Aufgabe `Gathering Facts` bestimmt und identifiziert die Betriebssystem-Distribution des verwalteten Hosts. Die Variable `supported_distros` wurde vom Autor des Playbooks erstellt und enthält eine Liste der Betriebssystem-Distributionen, die vom Playbook unterstützt werden. Wenn sich der Wert von `ansible_distribution` in der Liste `supported_distros` befindet, ist die Bedingung erfüllt und die Aufgabe wird ausgeführt.

```
---
```

```
- name: Demonstrate the "in" keyword
  hosts: all
  gather_facts: yes
  vars:
    supported_distros:
```

```

- RedHat
- Fedora
tasks:
  - name: Install httpd using yum, where supported
    yum:
      name: http
      state: present
    when: ansible_distribution in supported_distros
  
```



### Wichtig

Beachten Sie die Einrückung der Anweisung `when`. Weil die Anweisung `when` keine Modulvariable ist, muss sie „außerhalb“ des Moduls positioniert werden, indem sie auf der obersten Ebene der Aufgabe eingerückt wird.

Eine Aufgabe ist ein YAML-Hash/-Dictionary und die Anweisung `when` ist einfach ein weiterer Schlüssel in der Aufgabe, ebenso wie der Name der Aufgabe und das Modul, das sie verwendet. Nach gängiger Konvention werden eventuell vorhandene Keywords vom Typ `when` nach dem Namen der Aufgabe und dem Modul (und den Modularumenten) positioniert.

## Testen von Mehrfachbedingungen

Die Anweisung `when` kann zur Auswertung mehrerer Bedingungen verwendet werden. Zu diesem Zweck können Bedingungen mit den Keywords `and` oder `or` kombiniert und mit Klammern gruppiert werden.

Die folgenden Ausschnitte zeigen einige Beispiele, wie Mehrfachbedingungen ausgedrückt werden können.

- Wenn eine bedingte Anweisung nur mit einer Bedingung erfüllt sein soll, sollten Sie die Anweisung `or` verwenden. Beispielsweise ist die folgende Bedingung erfüllt, wenn auf dem Rechner Red Hat Enterprise Linux oder Fedora ausgeführt wird:

```
when: ansible_distribution == "RedHat" or ansible_distribution == "Fedora"
```

- Mit der Operation `and` müssen beide Bedingungen für die gesamte Bedingungsanweisung erfüllt sein. Beispielsweise wird die folgende Bedingung erfüllt, wenn der Remote-Host der Red Hat Enterprise Linux 7.5-Host und der installierte Kernel die angegebene Version ist:

```
when: ansible_distribution_version == "7.5" and ansible_kernel ==
      "3.10.0-327.el7.x86_64"
```

Das Keyword `when` unterstützt zudem die Verwendung einer Liste zum Beschreiben einer Liste der Bedingungen. Wenn eine Liste für das Keyword `when` bereitgestellt wird, werden alle Bedingungen mit der Funktion `and` kombiniert. Im folgenden Beispiel wird eine andere Möglichkeit gezeigt, wie mehrere Bedingungsanweisungen mit dem Operator `and` kombiniert werden können:

```

when:
  - ansible_distribution_version == "7.5"
  - ansible_kernel == "3.10.0-327.el7.x86_64"
  
```

Dieses Format verbessert die Lesbarkeit, ein Hauptziel von gut geschriebenen Ansible Playbooks.

- Komplexere bedingte Anweisungen können ausgedrückt werden, indem Bedingungen in Klammern gruppiert werden. Dies stellt sicher, dass sie richtig interpretiert werden.
- Beispielsweise wird die folgende bedingte Anweisung erfüllt, wenn auf dem Rechner Red Hat Enterprise Linux 7 oder Fedora 28 ausgeführt wird. In diesem Beispiel wird das Größer-als-Zeichen (>) verwendet, damit die lange Bedingung im Playbook zum Erleichtern des Lesens auf mehrere Zeilen aufgeteilt werden kann.

```
when: >
  ( ansible_distribution == "RedHat" and
    ansible_distribution_major_version == "7" )
  or
  ( ansible_distribution == "Fedora" and
    ansible_distribution_major_version == "28" )
```

## Kombinieren von Loops und bedingten Aufgaben

Sie können Loops und Bedingungen kombinieren.

Im folgenden Beispiel wird das Paket `mariadb-server` durch das Modul `yum` installiert, wenn ein Dateisystem auf / mit mehr als 300 MB freiem Speicher gemountet ist. Der Fakt `ansible_mounts` ist eine Liste von Dictionaries, wobei jedes Informationen über ein angeschlossenes Dateisystem repräsentiert. Der Loop durchläuft jedes Dictionary in der Liste und die Bedingungsanweisung ist erst erfüllt, wenn ein Dictionary gefunden wird, welches ein eingehängtes Dateisystem repräsentiert, das beide Bedingungen erfüllt.

```
- name: install mariadb-server if enough space on root
  yum:
    name: mariadb-server
    state: latest
  loop: "{{ ansible_mounts }}"
  when: item.mount == "/" and item.size_available > 3000000000
```



### Wichtig

Wenn Sie `when` mit `loop` für eine Aufgabe verwenden, wird die `when` - Anweisung für jedes Element geprüft.

Hier finden Sie ein weiteres Beispiel, in dem Bedingungen und Registerwerte kombiniert werden. Das folgende, kommentierte Playbook startet den Service `httpd` nur dann neu, wenn der Service `postfix` läuft:

```
---
- name: Restart HTTPD if Postfix is Running
  hosts: all
  tasks:
    - name: Get Postfix server status
      command: /usr/bin/systemctl is-active postfix ①
      ignore_errors: yes ②
```

```
register: result③

- name: Restart Apache HTTPD based on Postfix status
  service:
    name: httpd
    state: restarted
  when: result.rc == 0④
```

- ① Läuft Postfix oder nicht?
- ② Sollte Postfix nicht laufen und der Befehl fehlschlagen, dann stoppen Sie den Vorgang nicht.
- ③ Speichert Informationen zu den Modulergebnissen in einer Variable mit dem Namen **result**.
- ④ Bewertet die Ausgabe der Postfix-Aufgabe. Wenn der Beendigungscode des Befehls `systemctl` dem Wert 0 entspricht, ist Postfix aktiv und diese Aufgabe startet den Service `httpd` neu.



## Literaturhinweise

### Loops – Ansible-Dokumentation

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_loops.html](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_loops.html)

### Tests – Ansible-Dokumentation

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_tests.html](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_tests.html)

### Conditionals – Ansible-Dokumentation

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_conditionals.html](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_conditionals.html)

### What Makes A Valid Variable Name – Variables – Ansible-Dokumentation

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_variables.html#what-makes-a-valid-variable-name](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_variables.html#what-makes-a-valid-variable-name)

## ► Angeleitete Übung

# Loops und Aufgaben mit Bedingungen schreiben

In dieser Übung schreiben Sie ein Playbook mit Aufgaben, die Bedingungen und Loops enthalten.

## Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Implementieren der Ansible-Bedingungen mit dem Keyword `when`.
- Implementieren der Aufgabeniteration mit dem Keyword `loop` in Verbindung mit Bedingungen.

## Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab control-flow start` aus. Dieses Skript erstellt das Arbeitsverzeichnis `/home/student/control-flow`.

```
[student@workstation ~]$ lab control-flow start
```

## Anweisungen

- 1. Wechseln Sie auf `workstation.lab.example.com` zum Projektverzeichnis `/home/student/control-flow`.

```
[student@workstation ~]$ cd ~/control-flow
[student@workstation control-flow]$
```

- 2. Das Übungsskript hat eine Ansible-Konfigurationsdatei sowie eine Inventardatei erstellt. Diese Inventardatei enthält den Server `servera.lab.example.com` in der Hostgruppe `database_dev` und den Server `serverb.lab.example.com` in der Hostgruppe `database_prod`. Prüfen Sie die Datei, bevor Sie fortfahren.

```
[student@workstation control-flow]$ cat inventory
[database_dev]
servera.lab.example.com

[database_prod]
serverb.lab.example.com
```

- 3. Erstellen Sie das Playbook `playbook.yml` das ein Play mit zwei Aufgaben enthält. Verwenden Sie die Hostgruppe `database_dev`. Die erste Aufgabe installiert die

## Kapitel 4 | Implementieren der Aufgabensteuerung

erforderlichen MariaDB-Pakete, und die zweite Aufgabe stellt sicher, dass der MariaDB-Service ausgeführt wird.

- 3.1. Öffnen Sie das Playbook in einem Texteditor. Definieren Sie die Variable `mariadb_packages` mit zwei Werten: `mariadb-server` und `python3-PyMySQL`. Das Playbook verwendet die Variable, um die erforderlichen Pakete zu installieren. Die Datei sollte wie folgt aussehen:

```
---
```

```
- name: MariaDB server is running
hosts: database_dev
vars:
  mariadb_packages:
    - mariadb-server
    - python3-PyMySQL
```

- 3.2. Definieren Sie eine Aufgabe, die das Modul `yum` und die Variable `mariadb_packages` verwendet. Die Aufgabe installiert die erforderlichen Pakete. Die Aufgabe sollte wie folgt lauten:

```
tasks:
- name: MariaDB packages are installed
  yum:
    name: "{{ item }}"
    state: present
  loop: "{{ mariadb_packages }}"
```

- 3.3. Definieren Sie eine zweite Aufgabe, um den Service `mariadb` zu starten. Die Aufgabe sollte wie folgt lauten:

```
- name: Start MariaDB service
  service:
    name: mariadb
    state: started
    enabled: true
```

- 4. Führen Sie das Playbook aus und sehen Sie sich die Ausgabe des Plays an.

```
[student@workstation control-flow]$ ansible-playbook playbook.yml

PLAY [MariaDB server is running] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [MariaDB packages are installed] ****
changed: [servera.lab.example.com] => (item=mariadb-server)
changed: [servera.lab.example.com] => (item=python3-PyMySQL)

TASK [Start MariaDB service] ****
changed: [servera.lab.example.com]
```

```
PLAY RECAP ****
servera.lab.example.com      : ok=3      changed=2      unreachable=0      failed=0
```

- 5. Aktualisieren Sie die erste Aufgabe so, dass sie nur ausgeführt wird, wenn der verwaltete Host Red Hat Enterprise Linux als Betriebssystem verwendet. Aktualisieren Sie das Play, um die Hostgruppe database\_prod zu verwenden. Die Aufgabe sollte wie folgt lauten:

```
- name: MariaDB server is running
hosts: database_prod
vars:
...output omitted...
tasks:
- name: MariaDB packages are installed
  yum:
    name: "{{ item }}"
    state: present
  loop: "{{ mariadb_packages }}"
  when: ansible_distribution == "RedHat"
```

- 6. Verifizieren Sie, dass die verwalteten Hosts in der Hostgruppe database\_prod Red Hat Enterprise Linux als Betriebssystem verwenden.

```
[student@workstation control-flow]$ ansible database_prod -m command \
> -a 'cat /etc/redhat-release' -u devops --become
serverb.lab.example.com | CHANGED | rc=0 >>
Red Hat Enterprise Linux release 8.4 (Ootpa)
```

- 7. Führen Sie das Playbook erneut aus und sehen Sie sich die Ausgabe des Plays an.

```
[student@workstation control-flow]$ ansible-playbook playbook.yml

PLAY [MariaDB server is running] ****
TASK [Gathering Facts] ****
ok: [serverb.lab.example.com]

TASK [MariaDB packages are installed] ****
changed: [serverb.lab.example.com] => (item=mariadb-server)
changed: [serverb.lab.example.com] => (item=python3-PyMySQL)

TASK [Start MariaDB service] ****
changed: [serverb.lab.example.com]

PLAY RECAP ****
serverb.lab.example.com      : ok=3      changed=2      unreachable=0      failed=0
```

Ansible führt die Aufgabe aus, da serverb.lab.example.com Red Hat Enterprise Linux verwendet.

## Beenden

Führen Sie auf `workstation` das Skript `lab control-flow finish` aus, um die in dieser Übung erstellten Ressourcen zu bereinigen.

```
[student@workstation ~]$ lab control-flow finish
```

Hiermit ist die angeleitete Übung beendet.

# Implementieren von Handlern

## Ziele

Nach Abschluss dieses Abschnitts sollten Sie in der Lage sein, eine Aufgabe zu implementieren, die nur ausgeführt wird, wenn eine andere Aufgabe den verwalteten Host ändert.

## Ansible-Handler

Ansible-Module sollen *idempotent* sein. Das bedeutet, dass in einem ordnungsgemäß geschriebenen Playbook, das Playbook und entsprechende Aufgaben mehrfach ausgeführt werden können, ohne den verwalteten Host zu verändern. Es sei denn, es muss eine Änderung stattfinden, um den verwalteten Host in einen gewünschten Status zu versetzen.

Manchmal, wenn eine Aufgabe eine Änderung am System vorgenommen hat, muss eine weitere Aufgabe ausgeführt werden. Beispielsweise könnte eine Änderung an der Konfigurationsdatei eines Services erfordern, dass der Service erneut geladen wird, sodass die geänderte Konfiguration in Kraft tritt.

Handlers sind Aufgaben, die auf eine Meldung reagieren, die von anderen Aufgaben ausgelöst wurde. Aufgaben benachrichtigen ihre Handler nur, wenn die Aufgabe etwas auf einem verwalteten Host ändert. Jeder Handler wird durch seinen Namen nach dem Aufgabenblock des Plays ausgelöst. Wenn keine Aufgabe den Handler nach Namen aufruft, wird er nicht ausgeführt. Wenn eine oder mehrere Aufgaben den Handler benachrichtigen, dann wird der Handler exakt einmal ausgeführt, nachdem alle anderen Aufgaben im Play abgeschlossen sind. Da Handler Aufgaben sind, können Administratoren die gleichen Module in Handlern verwenden, die sie in anderen Aufgaben verwenden. Normalerweise werden Handler zum erneuten Hochfahren von Hosts oder zum Neustart von Services genutzt.



### Anmerkung

Verwenden Sie eindeutige Namen für Ihre Handler. Wenn mehrere Handler mit dem gleichen Namen definiert sind, wird nur der letzte Handler, der mit dem gemeinsamen Namen definiert wurde, ausgeführt.

Handlers können als *inaktive* Aufgaben angesehen werden, die nur ausgelöst werden, wenn sie explizit mit der Anweisung `notify` aufgerufen werden. Der folgende Auszug zeigt, wie der Apache-Server nur dann vom Handler `restart apache` neu gestartet wird, wenn eine Konfigurationsdatei aktualisiert und dies gemeldet wird:

```
tasks:  
  - name: copy demo.example.conf configuration template①  
    template:  
      src: /var/lib/templates/demo.example.conf.template  
      dest: /etc/httpd/conf.d/demo.example.conf  
    notify: ②  
      - restart apache③  
  
handlers: ④
```

```
- name: restart apache5
  service:
    name: httpd
    state: restarted
```

- ① Die Aufgabe, die den Handler benachrichtigt.
- ② Die Anweisung `notify` deutet darauf hin, dass die Aufgabe einen Handler auslösen muss.
- ③ Der Name des auszuführenden Handlers.
- ④ Das Keyword `handlers` gibt den Start der Liste der Handler-Aufgaben an.
- ⑤ Der Name des Handlers, der von den Aufgaben aufgerufen wird.
- ⑥ Das vom Handler zu verwendende Modul.

Im vorgenannten Beispiel wird der Handler `restart apache` ausgelöst, wenn die Aufgabe `template` meldet, dass eine Änderung stattfand. Eine Aufgabe kann mehr als einen Handler im zugehörigen Abschnitt `notify` benachrichtigen. Ansible behandelt die Anweisung `notify` wie ein Array und iteriert über die Handler-Namen:

```
tasks:
  - name: copy demo.example.conf configuration template
    template:
      src: /var/lib/templates/demo.example.conf.template
      dest: /etc/httpd/conf.d/demo.example.conf
    notify:
      - restart mysql
      - restart apache

handlers:
  - name: restart mysql
    service:
      name: mariadb
      state: restarted

  - name: restart apache
    service:
      name: httpd
      state: restarted
```

## Beschreiben der Vorteile der Verwendung von Handlern

Wie in der Ansible-Dokumentation beschrieben, sind zur Verwendung von Handlern einige wichtige Informationen zu beachten:

- Handler werden immer in der Reihenfolge ausgeführt, die im Abschnitt `handlers` des Plays angegebenen ist. Sie werden weder in der Reihenfolge ausgeführt, in der sie von Anweisungen des Typs `notify` in einer Aufgabe aufgelistet werden, noch in der Reihenfolge, in der sie von den Aufgaben benachrichtigt werden.
- Handler werden normalerweise erst ausgeführt, wenn alle anderen Aufgaben im Play abgeschlossen sind. Ein Handler, der von einer Aufgabe im Abschnitt `tasks` des Playbooks

aufgerufen wird, wird erst ausgeführt, wenn *alle* Aufgaben unter `tasks` bearbeitet wurden. (Es gibt einige kleinere Ausnahmen.)

- Handler-Namen gehören zu einem Play-basierten Namespace. Wenn zwei Handler fälschlicherweise den gleichen Namen haben, wird nur einer ausgeführt.
- Sogar wenn mehr als eine Aufgabe den Handler benachrichtigt, wird der Handler nur einmal ausgeführt. Wenn es keine Benachrichtigungen seitens der Aufgaben gibt, wird kein Handler ausgeführt.
- Wenn eine Aufgabe, die eine Anweisung vom Typ `notify` enthält, kein Ergebnis vom Typ `changed` meldet (beispielsweise ist ein Paket bereits installiert und die Aufgabe meldet `ok`), wird der Handler nicht benachrichtigt. Der Handler wird übersprungen, sofern eine andere Aufgabe diesen nicht auslöst. Ansible benachrichtigt Handler nur dann, wenn eine Aufgabe den Status `changed` meldet.



### Wichtig

Handler sollen eine zusätzliche Aktion ausführen, wenn eine Aufgabe eine Änderung an einem verwalteten Host vornimmt. Sie sollten nicht als Ersatz für normale Aufgaben verwendet werden.



### Literaturhinweise

#### Intro to Playbooks – Ansible-Dokumentation

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_intro.html](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_intro.html)

## ► Angeleitete Übung

# Implementieren von Handlern

In dieser Übung werden Sie Handler in Playbooks implementieren.

## Ergebnisse

Sie sollten in der Lage sein, Handler in Playbooks zu definieren und sie bei Konfigurationsänderungen zu benachrichtigen.

## Bevor Sie Beginnen

Führen Sie `lab control-handlers start` auf `workstation` aus, um die Umgebung für die Übung zu konfigurieren. Dieses Skript erstellt das Projektverzeichnis `/home/student/control-handlers` und lädt die Ansible-Konfigurationsdatei und die Hostinventardatei herunter, die für die Übung erforderlich sind. Das Projektverzeichnis enthält zudem das teilweise vollständige Playbook `configure_db.yml`.

```
[student@workstation ~]$ lab control-handlers start
```

## Anweisungen

- 1. Öffnen Sie auf `workstation.lab.example.com` ein neues Terminal, und wechseln Sie zum Projektverzeichnis `/home/student/control-handlers`.

```
[student@workstation ~]$ cd ~/control-handlers  
[student@workstation control-handlers]$
```

- 2. Verwenden Sie in diesem Verzeichnis einen Texteditor, um die Playbook-Datei `configure_db.yml` zu bearbeiten. Dieses Playbook installiert und konfiguriert einen Datenbankserver. Wenn sich die Datenbankserverkonfiguration ändert, löst das Playbook einen Neustart des Datenbankservices aus und konfiguriert das Administratorpasswort für die Datenbank.

- 2.1. Überprüfen Sie das Playbook `configure_db.yml` in einem Texteditor. Es beginnt mit der Initialisierung einiger Variablen:

```
---  
- name: MariaDB server is installed  
hosts: databases  
vars:  
  db_packages: ①  
    - mariadb-server  
    - python3-PyMySQL  
  db_service: mariadb ②  
  resources_url: http://materials.example.com/labs/control-handlers ③
```

```
config_file_url: "{{ resources_url }}/my.cnf.standard" ④
config_file_dst: /etc/my.cnf ⑤
tasks:
```

- ① db\_packages definiert den Namen der Pakete, die für den Datenbankservice installiert werden sollen.
  - ② db\_service definiert den Namen des Datenbankservices.
  - ③ resources\_url repräsentiert die URL für das Ressourcenverzeichnis, in dem sich die Remote-Konfigurationsdateien befinden.
  - ④ config\_file\_url repräsentiert die URL der zu installierenden Datenbankkonfigurationsdatei.
  - ⑤ config\_file\_dst: Speicherort der installierten Konfigurationsdatei auf den verwalteten Hosts.
- 2.2. Definieren Sie in der Datei `configure_db.yml` eine Aufgabe, die das Modul `yum` verwendet, um die erforderlichen Datenbankpakete zu installieren, die in der Variablen `db_packages` definiert sind. Wenn die Aufgabe das System ändert, wird die Datenbank nicht installiert und Sie müssen den Handler `set db password` zum Festlegen Ihres anfänglichen Datenbankbenutzers und -passworts benachrichtigen. Beachten Sie, dass die Handler-Aufgabe, bei Benachrichtigung, erst ausgeführt wird, nachdem alle Aufgaben im Abschnitt `tasks` ausgeführt wurden.

Die Aufgabe sollte wie folgt lauten:

```
tasks:
- name: "{{ db_packages }} packages are installed"
  yum:
    name: "{{ db_packages }}"
    state: present
  notify:
    - set db password
```

### 2.3. 2.3)

Fügen Sie eine Aufgabe zum Starten hinzu und aktivieren Sie den Datenbankservice. Die Aufgabe sollte wie folgt laufen:

```
- name: Make sure the database service is running
  service:
    name: "{{ db_service }}"
    state: started
    enabled: true
```

### 2.4. Fügen Sie eine Aufgabe hinzu, die `my.cnf.standard` nach `/etc/my.cnf` auf dem verwalteten Host herunterlädt, und verwenden Sie dazu das Modul `get_url`. Fügen Sie eine Bedingung hinzu, die den Handler `restart db_service` benachrichtigt, um den Datenbankservice nach der Änderung einer Konfigurationsdatei neu zu starten. Die Aufgabe sollte folgendermaßen aussehen:

```

- name: The {{ config_file_dst }} file has been installed
  get_url:
    url: "{{ config_file_url }}"
    dest: "{{ config_file_dst }}"
    owner: mysql
    group: mysql
    force: yes
  notify:
    - restart db service

```

- 2.5. Fügen Sie das Keyword `handlers` hinzu, um den Start der Handler-Aufgaben zu definieren. Definieren Sie den ersten Handler `restart db service`, der den Service `mariadb` neu startet. Das Ergebnis sollte folgendermaßen aussehen:

```

handlers:
- name: restart db service
  service:
    name: "{{ db_service }}"
    state: restarted

```

- 2.6. Definieren Sie den zweiten Handler `set db password`, der das Administratorpasswort für den Datenbankservice festlegt. Der Handler verwendet das Modul `mysql_user`, um den Befehl auszuführen. Der Handler sollte wie folgt lauten:

```

- name: set db password
  mysql_user:
    name: root
    password: redhat

```

Wenn Sie fertig sind, sollte das Playbook wie folgt aussehen:

```

---
- name: MariaDB server is installed
  hosts: databases
  vars:
    db_packages:
      - mariadb-server
      - python3-PyMySQL
    db_service: mariadb
    resources_url: http://materials.example.com/labs/control-handlers
    config_file_url: "{{ resources_url }}/my.cnf.standard"
    config_file_dst: /etc/my.cnf
  tasks:
    - name: "{{ db_packages }} packages are installed"
      yum:
        name: "{{ db_packages }}"
        state: present
      notify:
        - set db password
    - name: Make sure the database service is running

```

```
service:
  name: "{{ db_service }}"
  state: started
  enabled: true

  - name: The {{ config_file_dst }} file has been installed
    get_url:
      url: "{{ config_file_url }}"
      dest: "{{ config_file_dst }}"
      owner: mysql
      group: mysql
      force: yes
    notify:
      - restart db service

handlers:
  - name: restart db service
    service:
      name: "{{ db_service }}"
      state: restarted

  - name: set db password
    mysql_user:
      name: root
      password: redhat
```

- 3. Verifizieren Sie vor Ausführung des Playbooks, dass dessen Syntax korrekt ist, indem Sie `ansible-playbook` mit der Option `--syntax-check` ausführen. Wenn Fehler gemeldet werden, korrigieren Sie sie, bevor Sie mit dem nächsten Schritt fortfahren. Es sollte eine Ausgabe ähnlich der folgenden angezeigt werden:

```
[student@workstation control-handlers]$ ansible-playbook configure_db.yml \
> --syntax-check

playbook: configure_db.yml
```

- 4. Führen Sie das Playbook `configure_db.yml` aus. Die Ausgabe zeigt an, dass die Handler ausgeführt werden.

```
[student@workstation control-handlers]$ ansible-playbook configure_db.yml

PLAY [Installing MariaDB server] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [['mariadb-server', 'python3-PyMySQL'] packages are installed] ****
changed: [servera.lab.example.com]

TASK [Make sure the database service is running] ****
changed: [servera.lab.example.com]

TASK [The /etc/my.cnf file has been installed] ****
```

```
changed: [servera.lab.example.com]

RUNNING HANDLER [restart db service] ****
changed: [servera.lab.example.com]

RUNNING HANDLER [set db password] ****
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=6    changed=5    unreachable=0    failed=0
```

- 5. Führen Sie das Playbook erneut aus.

```
[student@workstation control-handlers]$ ansible-playbook configure_db.yml

PLAY [Installing MariaDB server] ****

TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [['mariadb-server', 'python3-PyMySQL'] packages are installed] ****
ok: [servera.lab.example.com]

TASK [Make sure the database service is running] ****
ok: [servera.lab.example.com]

TASK [The /etc/my.cnf file has been installed] ****
ok: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=4    changed=0    unreachable=0    failed=0
```

Diesmal werden die Handlers übersprungen. Falls die Remote-Konfigurationsdatei künftig geändert wird, würde die Ausführung des Playbooks den Handler `restart db service`, nicht aber das Handler `set db password` auslösen.

## Beenden

Führen Sie auf `workstation` das Skript `lab control-handlers finish` aus, um die in dieser Übung erstellten Ressourcen zu bereinigen.

```
[student@workstation ~]$ lab control-handlers finish
```

Hiermit ist die angeleitete Übung beendet.

# Task-Fehler behandeln

## Ziele

Nach Abschluss dieses Abschnitts sollten Kursteilnehmer in der Lage sein, festzulegen was passiert wenn eine Aufgabe fehlschlägt und unter welchen Bedingungen eine Aufgabe fehlschlägt.

## Verwalten von Aufgabenfehlern in Plays

Ansible wertet die Rückgabecodes der einzelnen Aufgaben aus, um festzustellen, ob die jeweilige Aufgabe erfolgreich durchgeführt wurde oder fehlgeschlagen ist. Wenn eine Aufgabe fehlschlägt, bricht Ansible normalerweise den Rest des Plays auf diesem Host sofort ab und überspringt alle nachfolgenden Aufgaben.

Manchmal jedoch möchten Sie die Ausführung des Plays vielleicht fortsetzen, selbst wenn eine Aufgabe fehlschlägt. Vielleicht erwarten Sie beispielsweise, dass eine bestimmte Aufgabe fehlgeschlagen könnte, und Sie möchten vielleicht fortfahren, indem Sie eine andere Aufgabe bedingt ausführen. Es gibt einige Ansible-Features, mit denen sich Aufgabenfehler besser handhaben lassen:

### Ignorieren fehlgeschlagener Aufgaben

Wenn eine Aufgabe fehlschlägt, wird das Play standardmäßig abgebrochen. Dieses Verhalten kann jedoch durch Ignorieren fehlgeschlagener Aufgaben außer Kraft gesetzt werden. Um dies zu erreichen, können Sie in einer Aufgabe das Schlüsselwort `ignore_errors` verwenden.

Der folgende Auszug zeigt, wie `ignore_errors` in einer Aufgabe verwendet wird, um die Ausführung des Playbooks fortzusetzen, selbst wenn die Aufgabe fehlschlägt. Wenn z. B. das Paket `notapkg` nicht existiert, dann schlägt das yum-Modul fehl. Wenn aber `ignore_errors` auf `yes` gesetzt ist, kann die Ausführung fortgesetzt werden.

```
- name: Latest version of notapkg is installed
  yum:
    name: notapkg
    state: latest
    ignore_errors: yes
```

### Erzwungene Ausführung von Handlern nach fehlgeschlagenen Aufgaben

Wenn eine Aufgabe fehlschlägt und das Play auf diesem Host abgebrochen wird, wird normalerweise kein Handler ausgeführt, der von vorherigen Aufgaben im Play benachrichtigt wurde. Wenn im Play das Keyword `force_handlers: yes` festgelegt wird, werden benachrichtigte Handler aufgerufen, selbst wenn das Play abgebrochen wurde, weil eine spätere Aufgabe fehlschlug.

Der folgende Auszug zeigt, wie das Keyword `force_handlers` in einem Play verwendet wird, um die Ausführung des Handlers zu erzwingen, selbst wenn eine Aufgabe fehlschlägt:

```

---
- hosts: all
  force_handlers: yes
  tasks:
    - name: a task which always notifies its handler
      command: /bin/true
      notify: restart the database

    - name: a task which fails because the package doesn't exist
      yum:
        name: notapkg
        state: latest

  handlers:
    - name: restart the database
      service:
        name: mariadb
        state: restarted

```



### Anmerkung

Denken Sie daran, dass Handler benachrichtigt werden, wenn eine Aufgabe ein `changed`-Ergebnis meldet, aber bei Meldung eines `ok`- oder `failed`-Ergebnisses nicht benachrichtigt werden.

## Angeben von Bedingungen für fehlgeschlagene Aufgaben

Mithilfe des Keyword `failed_when` können Sie in einer Aufgabe angeben, welche Bedingungen anzeigen, dass die Aufgabe fehlgeschlagen ist. Diese Möglichkeit wird häufig bei Modulen zur Ausführung von Befehlen genutzt. Diese Module führen einen Befehl möglicherweise erfolgreich aus, aber die Ausgabe des Befehls deutet auf einen Fehler hin.

Sie können z. B. ein Skript ausführen, das eine Fehlermeldung herausgibt, und diese Meldung nutzen, um den „`failed`“-Status für diese Aufgabe zu definieren. Der folgende Auszug zeigt, wie das Keyword `failed_when` in einer Aufgabe verwendet werden kann:

```

tasks:
  - name: Run user creation script
    shell: /usr/local/bin/create_users.sh
    register: command_result
    failed_when: "'Password missing' in command_result.stdout"

```

Das Modul `fail` kann auch zum Erzwingen eines Taskfehlers verwendet werden. Das obige Szenario kann alternativ in zwei Aufgaben erfasst werden:

```

tasks:
  - name: Run user creation script
    shell: /usr/local/bin/create_users.sh
    register: command_result
    ignore_errors: yes

  - name: Report script failure

```

```
fail:
  msg: "The password is missing in the output"
  when: "'Password missing' in command_result.stdout"
```

Sie können das Modul `fail` verwenden, um eine klare Fehlermeldung für die Aufgabe bereitzustellen. Dieser Ansatz ermöglicht auch einen verzögerten Fehler, sodass Sie Zwischenaufgaben ausführen können, um andere Änderungen abzuschließen oder rückgängig zu machen.

## Angeben des Zeitpunkts, zu dem eine Aufgabe „Changed“-Ergebnisse meldet

Wenn eine Aufgabe eine Änderung an einem verwalteten Host vornimmt, meldet sie den `changed`-Status und benachrichtigt Handler. Wenn eine Aufgabe keine Änderung vornehmen muss, meldet sie `ok` und benachrichtigt keine Handler.

Mithilfe des Keywords `changed_when` kann gesteuert werden, wann eine Aufgabe meldet, dass eine Änderung vorgenommen wurde. Das Modul `shell` im nächsten Beispiel etwa wird verwendet, um Kerberos-Anmelddaten zu erhalten, die von nachfolgenden Aufgaben verwendet werden. Diese Aufgabe würde normalerweise immer `changed` melden, wenn sie ausgeführt wird. Um dies zu unterdrücken, wird `changed_when: false` festgelegt, damit nur `ok` oder `failed` gemeldet wird.

```
- name: get Kerberos credentials as "admin"
  shell: echo "{{ krb_admin_pass }}" | kinit -f admin
  changed_when: false
```

Im Folgenden ein weiteres Beispiel mit dem `shell`-Modul, das `changed` auf Basis der Modulausgabe meldet, die von einer registrierten Variable erfasst wird:

```
tasks:
  - shell:
      cmd: /usr/local/bin/upgrade-database
    register: command_result
    changed_when: "'Success' in command_result.stdout"
    notify:
      - restart_database

handlers:
  - name: restart_database
    service:
      name: mariadb
      state: restarted
```

## Ansible-Blöcke und Fehlerbehebung

In Playbooks nutzt man `blocks` als Klauseln zur logischen Gruppierung von Aufgaben, mit denen gesteuert werden kann, wie Aufgaben ausgeführt werden. Ein Aufgabenblock kann z. B. ein `when`-Keyword enthalten, um eine Bedingung auf mehrere Aufgaben anzuwenden:

```
- name: block example
  hosts: all
  tasks:
```

**Kapitel 4 |** Implementieren der Aufgabensteuerung

```
- name: installing and configuring Yum versionlock plugin
  block:
    - name: package needed by yum
      yum:
        name: yum-plugin-versionlock
        state: present
    - name: lock version of tzdata
      lineinfile:
        dest: /etc/yum/pluginconf.d/versionlock.list
        line: tzdata-2016j-1
        state: present
  when: ansible_distribution == "RedHat"
```

In Verbindung mit den Anweisungen `rescue` und `always` ermöglichen Blöcke außerdem eine Fehlerbehandlung. Wenn eine Aufgabe in einem Block fehlschlägt, werden nacheinander die Aufgaben im zugehörigen `rescue`-Block ausgeführt, um fortfahren zu können. Nachdem die Aufgaben in der Blockklausel sowie, falls ein Fehler aufgetreten ist, die Aufgaben in der Rettungsklausel ausgeführt wurden, werden Aufgaben mit der `always`-Klausel ausgeführt. Zusammengefasst:

- `block`: Definiert die auszuführenden Hauptaufgaben.
- `rescue`: Definiert die Aufgaben, die ausgeführt werden, wenn die Aufgaben in `block` fehlschlagen.
- `always`: Definiert die Aufgaben, die immer unabhängig vom Erfolg oder Misserfolg der Aufgaben ausgeführt werden, die in `block` und `rescue` definiert sind.

Das folgende Beispiel zeigt die Implementierung eines Blocks in einem Playbook. Auch wenn die in `block` definierten Aufgaben fehlschlagen, werden die Aufgaben ausgeführt, die in den Klauseln `rescue` und `always` definiert sind.

```
tasks:
  - name: Upgrade DB
    block:
      - name: upgrade the database
        shell:
          cmd: /usr/local/lib/upgrade-database
    rescue:
      - name: revert the database upgrade
        shell:
          cmd: /usr/local/lib/revert-database
    always:
      - name: always restart the database
        service:
          name: mariadb
          state: restarted
```

Die `when`-Bedingung in einer `block`-Klausel gilt auch für die zugehörigen Klauseln `rescue` und `always`, falls vorhanden.



### Literaturhinweise

#### Error Handling in Playbooks – Ansible-Dokumentation

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_error\\_handling.html](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_error_handling.html)

#### Error Handling – Blocks – Ansible-Dokumentation

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_blocks.html#blocks-error-handling](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_blocks.html#blocks-error-handling)

## ► Angeleitete Übung

# Task-Fehler behandeln

In dieser Übung werden Sie verschiedene Möglichkeiten zur Behandlung von Aufgabenfehlern in einem Ansible-Playbook kennenlernen.

## Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Fehlgeschlagene Befehle während der Playbook-Ausführung ignorieren.
- Ausführung von Handlers forcieren.
- Die Fehlerursache in Aufgaben überschreiben.
- Status changed für Aufgaben überschreiben.
- Implementieren Sie `block`, `rescue` und `always` in Playbooks.

## Bevor Sie Beginnen

Führen Sie auf `workstation` das Übungs-Start-Skript aus, um zu prüfen, ob die Umgebung für die Übung bereit ist. Dieses Skript erstellt das Arbeitsverzeichnis `/home/student/control-errors`.

```
[student@workstation ~]$ lab control-errors start
```

## Anweisungen

- 1. Wechseln Sie auf `workstation.lab.example.com` zum Projektverzeichnis `/home/student/control-errors`.

```
[student@workstation ~]$ cd ~/control-errors  
[student@workstation control-errors]$
```

- 2. Das Übungsskript erstellte eine Ansible-Konfigurationsdatei sowie eine Inventardatei, die den Server `servera.lab.example.com` in der Gruppe `databases` enthält. Prüfen Sie die Datei, bevor Sie fortfahren.

- 3. Erstellen Sie das Playbook `playbook.yml` das ein Play mit zwei Aufgaben enthält. Erstellen Sie die erste Aufgabe mit einem gewollten Fehler, damit sie fehlschlägt.

- 3.1. Öffnen Sie das Playbook in einem Texteditor. Definieren Sie drei Variablen: `web_package` mit dem Wert `http`, `db_package` mit dem Wert `mariadb-server` und `db_service` mit dem Wert `mariadb`. Diese Variablen werden verwendet, um die erforderlichen Pakete zu installieren und den Server zu starten.

Der Wert `http` ist ein beabsichtigter Fehler im Paketnamen. Die Datei sollte wie folgt aussehen:

```
---  
- name: Task Failure Exercise  
hosts: databases  
vars:  
  web_package: http  
  db_package: mariadb-server  
  db_service: mariadb
```

- 3.2. Definieren Sie zwei Aufgaben, die das Modul yum und die zwei Variablen web\_package und db\_package verwenden. Mithilfe dieser Aufgaben werden die erforderlichen Pakete installiert. Die Aufgaben sollten folgendermaßen aussehen:

```
tasks:  
  - name: Install {{ web_package }} package  
    yum:  
      name: "{{ web_package }}"  
      state: present  
  
  - name: Install {{ db_package }} package  
    yum:  
      name: "{{ db_package }}"  
      state: present
```

- 4. Führen Sie das Playbook aus und sehen Sie sich die Ausgabe des Plays an.

```
[student@workstation control-errors]$ ansible-playbook playbook.yml  
  
PLAY [Task Failure Exercise] *****  
  
TASK [Gathering Facts] *****  
ok: [servera.lab.example.com]  
  
TASK [Install http package] *****  
fatal: [servera.lab.example.com]: FAILED! => {"changed": false, "failures":  
  ["No package http available."], "msg": "Failed to install some of the specified  
  packages", "rc": 1, "results": []}  
  
PLAY RECAP *****  
servera.lab.example.com : ok=1    changed=0    unreachable=0    failed=1
```

Die Aufgabe schlug fehl, weil kein Paket mit dem Namen http vorhanden ist. Da die erste Aufgabe fehlschlug, wurde die zweite Aufgabe nicht ausgeführt.

- 5. Aktualisieren Sie die erste Aufgabe so, dass Fehler ignoriert werden, indem Sie das Keyword ignore\_errors hinzufügen. Die Aufgaben sollten folgendermaßen aussehen:

```
tasks:  
  - name: Install {{ web_package }} package  
    yum:  
      name: "{{ web_package }}"  
      state: present  
      ignore_errors: yes
```

```
- name: Install {{ db_package }} package
  yum:
    name: "{{ db_package }}"
    state: present
```

- 6. Führen Sie das Playbook erneut aus und sehen Sie sich die Ausgabe des Plays an.

```
[student@workstation control-errors]$ ansible-playbook playbook.yml

PLAY [Task Failure Exercise] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Install http package] ****
fatal: [servera.lab.example.com]: FAILED! => {"changed": false, "failures": [
    {"No package http available."}, "msg": "Failed to install some of the specified
packages", "rc": 1, "results": []}
...ignoring

TASK [Install mariadb-server package] ****
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com      : ok=3      changed=1      unreachable=0      failed=0
```

Obwohl die erste Aufgabe fehlschlug, führte Ansible die zweite aus.

- 7. In diesem Schritt richten Sie ein **block**-Keyword so ein, dass Sie mit den Funktionsweisen experimentieren können.

- 7.1. Aktualisieren Sie das Playbook, indem Sie die erste Aufgabe in eine **block**-Klausel einbetten. Entfernen Sie die Zeile, in der `ignore_errors: yes` festgelegt wird. Der Block sollte folgendermaßen aussehen:

```
- name: Attempt to set up a webserver
  block:
    - name: Install {{ web_package }} package
      yum:
        name: "{{ web_package }}"
        state: present
```

- 7.2. Betten Sie die Aufgabe, mit der das *mariadb-server*-Paket installiert wird, in eine **rescue**-Klausel ein. Die Aufgabe wird ausgeführt, wenn die in der **block**-Klausel aufgeführte Aufgabe fehlschlägt. Die **Block**-Klausel sollte folgendermaßen aussehen:

```
rescue:
  - name: Install {{ db_package }} package
    yum:
      name: "{{ db_package }}"
      state: present
```

- 7.3. Fügen Sie schließlich eine `always`-Klausel hinzu, um den Datenbankserver bei der Installation mit dem Modul `service` zu installieren. Die Klausel sollte folgendermaßen aussehen:

```
always:
  - name: Start {{ db_service }} service
    service:
      name: "{{ db_service }}"
      state: started
```

- 7.4. Der Aufgabenabschnitt sollte wie folgt lauten:

```
tasks:
  - name: Attempt to set up a webserver
    block:
      - name: Install {{ web_package }} package
        yum:
          name: "{{ web_package }}"
          state: present
    rescue:
      - name: Install {{ db_package }} package
        yum:
          name: "{{ db_package }}"
          state: present
  always:
    - name: Start {{ db_service }} service
      service:
        name: "{{ db_service }}"
        state: started
```

- 8. Starten Sie das Playbook erneut und beobachten Sie die Ausgabe.

- 8.1. Führen Sie das Playbook aus. Die Aufgabe in dem Block stellt sicher, dass das installierte `web_package` fehlschlägt. Dies hat zur Folge, dass die Aufgabe im `rescue`-Block ausgeführt wird. Anschließend wird die Aufgabe im `always`-Block ausgeführt.

```
[student@workstation control-errors]$ ansible-playbook playbook.yml

PLAY [Task Failure Exercise] ****

TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Install http package] ****
fatal: [servera.lab.example.com]: FAILED! => {"changed": false, "failures": ["No package http available."], "msg": "Failed to install some of the specified packages", "rc": 1, "results": []}

TASK [Install mariadb-server package] ****
ok: [servera.lab.example.com]

TASK [Start mariadb service] ****
```

**Kapitel 4 |** Implementieren der Aufgabensteuerung

```
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=3    changed=1    unreachable=0    failed=1
```

- 8.2. Bearbeiten Sie das Playbook und korrigieren Sie den Wert der Variable `web_package` auf `httpd`. Damit wird die Aufgabe im Block erfolgreich abgeschlossen, wenn das Playbook das nächste Mal ausgeführt wird.

```
vars:
  web_package: httpd
  db_package: mariadb-server
  db_service: mariadb
```

- 8.3. Führen Sie das Playbook erneut aus. Diesmal schlägt die Aufgabe im Block nicht fehl. Daher wird die Aufgabe im `rescue`-Abschnitt ignoriert. Die Aufgabe im `always`-Block wird nach wie vor ausgeführt.

```
[student@workstation control-errors]$ ansible-playbook playbook.yml

PLAY [Task Failure Exercise] ****

TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Install httpd package] ****
changed: [servera.lab.example.com]

TASK [Start mariadb service] ****
ok: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=3    changed=1    unreachable=0    failed=0
```

- 9. In diesem Schritt wird untersucht, wie Sie die Bedingung steuern können, die dazu führt, dass eine Aufgabe als "changed" an den verwalteten Host gemeldet wird.

- 9.1. Bearbeiten Sie das Playbook und fügen Sie am Anfang des Plays vor dem `block` zwei Aufgaben hinzu. Die erste Aufgabe verwendet das Modul `command`, um den Befehl `date` auszuführen und das Ergebnis in der Variable `command_result` zu registrieren. Die zweite Aufgabe verwendet das Modul `debug`, um die Standardausgabe des Befehls der ersten Aufgabe zu drucken.

```
tasks:
  - name: Check local time
    command: date
    register: command_result

  - name: Print local time
    debug:
      var: command_result.stdout
```

- 9.2. Führen Sie das Playbook aus. Sie sollten feststellen, dass die erste Aufgabe, die das Modul command ausführt, changed meldet, obwohl keine Änderungen am Remote-System vorgenommen, sondern nur Informationen über die Uhrzeit gesammelt wurden. Grund hierfür ist, dass das Modul command nicht den Unterschied zwischen einem Befehl zur Datensammlung und einem Befehl, der den Status ändert, erkennen kann.

```
[student@workstation control-errors]$ ansible-playbook playbook.yml

PLAY [Task Failure Exercise] *****

TASK [Gathering Facts] *****
ok: [servera.lab.example.com]

TASK [Check local time] *****
changed: [servera.lab.example.com]

TASK [Print local time] *****
ok: [servera.lab.example.com] => {
    "command_result.stdout": "mié mar 27 08:07:08 EDT 2019"
}

TASK [Install httpd package] *****
ok: [servera.lab.example.com]

TASK [Start mariadb service] *****
ok: [servera.lab.example.com]

PLAY RECAP *****
servera.lab.example.com      : ok=5      changed=1      unreachable=0      failed=0
```

Wenn Sie das Playbook erneut ausführen, meldet die Aufgabe Check local time wieder changed.

- 9.3. Diese command-Aufgabe sollte nicht bei jeder Ausführung changed melden, weil sie keine Änderungen am verwalteten Host vornimmt. Da Sie wissen, dass die Aufgabe niemals Änderungen an einem verwalteten Host vornimmt, fügen Sie die Zeile changed\_when: false zur Aufgabe hinzu, um die „changed“-Meldung zu unterdrücken.

```
tasks:
  - name: Check local time
    command: date
    register: command_result
    changed_when: false

  - name: Print local time
    debug:
      var: command_result.stdout
```

- 9.4. Führen Sie das Playbook erneut aus. Sie können feststellen, dass die Aufgabe nun ok meldet, aber nach wie vor ausgeführt wird und weiterhin die Uhrzeit in der Variable speichert.

```
[student@workstation control-errors]$ ansible-playbook playbook.yml

PLAY [Task Failure Exercise] ****

TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Check local time] ****
ok: [servera.lab.example.com]

TASK [Print local time] ****
ok: [servera.lab.example.com] => {
    "command_result.stdout": "mié mar 27 08:08:36 EDT 2019"
}

TASK [Install httpd package] ****
ok: [servera.lab.example.com]

TASK [Start mariadb service] ****
ok: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com      : ok=5    changed=0    unreachable=0    failed=0
```

- 10. Bearbeiten Sie als abschließende Übung das Playbook, um zu untersuchen, wie das Keyword `failed_when` mit Aufgaben interagiert.

- 10.1. Bearbeiten Sie die Aufgabe `Install {{ web_package }} package` so, dass sie mit „failed“ einen Fehlschlag meldet, wenn `web_package` den Wert `httpd` aufweist. Da dies der Fall ist, meldet die Aufgabe einen Fehlschlag, wenn Sie das Play ausführen.

Achten Sie auf Ihre Einrückungen, um sicherzustellen, dass das Keyword korrekt in der Aufgabe platziert ist.

```
- block:
  - name: Install {{ web_package }} package
    yum:
      name: "{{ web_package }}"
      state: present
    failed_when: web_package == "httpd"
```

- 10.2. Führen Sie das Playbook aus.

```
[student@workstation control-errors]$ ansible-playbook playbook.yml

PLAY [Task Failure Exercise] ****

TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Check local time] ****
ok: [servera.lab.example.com]
```

```
TASK [Print local time] ****
ok: [servera.lab.example.com] => {
    "command_result.stdout": "mié mar 27 08:09:35 EDT 2019"
}

TASK [Install httpd package] ****
fatal: [servera.lab.example.com]: FAILED! => {"changed": false,
  "failed_when_result": true, "msg": "Nothing to do", "rc": 0, "results": [
  {"Installed: httpd"}]

TASK [Install mariadb-server package] ****
ok: [servera.lab.example.com]

TASK [Start mariadb service] ****
ok: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com      : ok=5      changed=0      unreachable=0      failed=1
```

Überprüfen Sie sorgfältig die Ausgabe. Die Aufgabe `Install httpd package` meldet, dass sie fehlgeschlagen ist, aber eigentlich wurde sie ausgeführt und hat zuerst sichergestellt, dass das Paket installiert ist. Mit dem Keyword `failed_when` wird der Status, den die Aufgabe meldet, geändert, *nachdem* die Aufgabe ausgeführt wurde, aber das Verhalten der Aufgabe selbst wird nicht geändert.

Durch den gemeldeten Fehlschlag kann sich jedoch das Verhalten des restlichen Plays ändern. Da die Aufgabe in einen Block eingebettet war und einen Fehlschlag meldete, wurde die Aufgabe `Install mariadb-server package` im rescue-Abschnitt des Blocks ausgeführt.

## Beenden

Führen Sie auf `workstation` das Skript `lab control-errors finish` aus, um die in dieser Übung erstellten Ressourcen zu bereinigen.

```
[student@workstation ~]$ lab control-errors finish
```

Hiermit ist die angeleitete Übung beendet.

## ► Praktische Übung

# Implementieren der Aufgabensteuerung

### Performance-Checkliste

In dieser Übung werden Sie den Apache-Webserver installieren und mit `mod_ssl` sichern. Sie verwenden Bedingungen, Handler und die Aufgabenfehlerbehandlung in Ihrem Playbook, um die Umgebung bereitzustellen.

### Ergebnisse

Sie sollten in der Lage sein, Bedingungen in Ansible Playbooks zu definieren, Loops einzurichten, die Elemente durchlaufen, Handler in Playbooks zu definieren und Aufgabenfehler zu behandeln.

### Bevor Sie Beginnen

Melden Sie sich auf `workstation` als Benutzer `student` an, und führen Sie `lab control-review start` aus. Dieses Skript stellt sicher, dass der verwaltete Host `serverb` im Netzwerk erreichbar ist. Damit wird auch die Installation der korrekten Ansible-Konfigurationsdatei und des korrekten Inventars auf dem Kontrollknoten gewährleistet.

```
[student@workstation ~]$ lab control-review start
```

### Anweisungen

1. Wechseln Sie auf `workstation` zum Projektverzeichnis `/home/student/control-review`.
2. Das Projektverzeichnis enthält das teilweise fertiggestellte Playbook `playbook.yml`. Fügen Sie in einem Texteditor eine Aufgabe hinzu, die das Modul `fail` unter dem Kommentar `#Fail Fast Message` verwendet. Geben Sie einen entsprechenden Namen für die Aufgabe an. Diese Aufgabe sollte nur ausgeführt werden, wenn das Remote-System die Mindestanforderungen nicht erfüllt.

Die Mindestanforderungen für den Remote-Host sind unten aufgeführt:

- Verfügt mindestens über die Menge an RAM, die von der Variablen `min_ram_mb` angegeben wird. Die Variable `min_ram_mb` ist in der Datei `vars.yml` definiert und hat einen Wert von 256.
  - Führt Red Hat Enterprise Linux aus.
3. Fügen Sie dem Playbook unter dem Kommentar `#Install all Packages` eine einzelne Aufgabe hinzu, um die neueste Version fehlender Pakete zu installieren. Erforderliche Pakete werden durch die Variable `packages` angegeben, die in der Datei `vars.yml` definiert ist. Der Aufgabename sollte `Ensure required packages are present` lauten.
  4. Fügen Sie dem Playbook unter dem Kommentar `#Enable and start services` eine einzelne Aufgabe hinzu, um alle Services zu starten. Alle von der Variablen `services` angegebenen Services, die in der Datei `vars.yml` definiert sind, sollten gestartet und aktiviert werden. Geben Sie einen entsprechenden Namen für die Aufgabe an.

5. Fügen Sie unter dem Kommentar `#Block of config tasks` dem Playbook einen Aufgabenblock hinzu. Dieser Block enthält zwei Aufgaben:
- Eine Aufgabe, um sicherzustellen, dass das in der Variable `ssl_cert_dir` angegebene Verzeichnis auf dem Remote-Host vorhanden ist. In diesem Verzeichnis werden die Zertifikate des Webservers gespeichert.
  - Eine Aufgabe, um alle von der Variable `web_config_files` angegebenen Dateien auf den Remote-Host zu kopieren. Überprüfen Sie die Struktur der Variablen `web_config_files` in der Datei `vars.yml`. Konfigurieren Sie die Aufgabe, um jede Datei in das richtige Zielverzeichnis auf dem Remote-Host zu kopieren.

Diese Aufgabe sollte den Handler `restart web service` auslösen, wenn eine dieser Dateien auf dem Remote-Server geändert wird.

Darüber hinaus wird eine Debug-Aufgabe ausgeführt, wenn eine der beiden obigen Aufgaben fehlschlägt. In diesem Fall gibt die Aufgabe die folgende Meldung aus: `One or more of the configuration changes failed, but the web service is still active.`

Geben Sie einen entsprechenden Namen für alle Aufgaben an.

6. Das Playbook konfiguriert den Remote-Host zum Überwachen von HTTPS-Standardanforderungen. Fügen Sie unter dem Kommentar `#Configure the firewall` dem Playbook eine einzelne Aufgabe hinzu, um `firewalld` zu konfigurieren.
- Diese Aufgabe sollte sicherstellen, dass der Remote-Host HTTP- und HTTPS-Standardverbindungen zulässt. Diese Konfigurationsänderungen sollten sofort wirksam sein und nach einem Neustart des Systems bestehen bleiben. Geben Sie einen entsprechenden Namen für die Aufgabe an.
7. Definieren Sie den Handler `restart web service`.
- Wenn diese Aufgabe ausgelöst wird, sollte sie den Webservice neu starten, der von der Variablen `web_service` definiert wird, die in der Datei `vars.yml` definiert ist.
8. Führen Sie im Projektverzeichnis `~/control-review` das Playbook `playbook.yml` aus. Das Playbook sollte fehlerfrei ausgeführt werden und die Ausführung der Handler-Aufgabe auslösen.
9. Verifizieren Sie, dass der Webserver nun mit einem selbstsignierten benutzerdefinierten Zertifikat auf HTTPS-Anforderungen reagiert, um die Verbindung zu verschlüsseln. Die Webserverantwort sollte der Zeichenfolge `Configured for both HTTP and HTTPS` entsprechen.

## Bewertung

Führen Sie auf `workstation` den Befehl `lab control-review grade` aus, um den Erfolg dieser Übung zu bestätigen. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie das Skript so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab control-review grade
```

## Beenden

Führen Sie den Befehl `lab control-review finish` aus, um nach der Übung eine Bereinigung durchzuführen.

```
[student@workstation ~]$ lab control-review finish
```

Hiermit ist die praktische Übung abgeschlossen.

## ► Lösung

# Implementieren der Aufgabensteuerung

### Performance-Checkliste

In dieser Übung werden Sie den Apache-Webserver installieren und mit `mod_ssl` sichern. Sie verwenden Bedingungen, Handler und die Aufgabenfehlerbehandlung in Ihrem Playbook, um die Umgebung bereitzustellen.

### Ergebnisse

Sie sollten in der Lage sein, Bedingungen in Ansible Playbooks zu definieren, Loops einzurichten, die Elemente durchlaufen, Handler in Playbooks zu definieren und Aufgabenfehler zu behandeln.

### Bevor Sie Beginnen

Melden Sie sich auf `workstation` als Benutzer `student` an, und führen Sie `lab control-review start` aus. Dieses Skript stellt sicher, dass der verwaltete Host `serverb` im Netzwerk erreichbar ist. Damit wird auch die Installation der korrekten Ansible-Konfigurationsdatei und des korrekten Inventars auf dem Kontrollknoten gewährleistet.

```
[student@workstation ~]$ lab control-review start
```

### Anweisungen

1. Wechseln Sie auf `workstation` zum Projektverzeichnis `/home/student/control-review`.

```
[student@workstation ~]$ cd ~/control-review
[student@workstation control-review]$
```

2. Das Projektverzeichnis enthält das teilweise fertiggestellte Playbook `playbook.yml`. Fügen Sie in einem Texteditor eine Aufgabe hinzu, die das Modul `fail` unter dem Kommentar `#Fail Fast Message` verwendet. Geben Sie einen entsprechenden Namen für die Aufgabe an. Diese Aufgabe sollte nur ausgeführt werden, wenn das Remote-System die Mindestanforderungen nicht erfüllt.

Die Mindestanforderungen für den Remote-Host sind unten aufgeführt:

- Verfügt mindestens über die Menge an RAM, die von der Variablen `min_ram_mb` angegeben wird. Die Variable `min_ram_mb` ist in der Datei `vars.yml` definiert und hat einen Wert von 256.
- Führt Red Hat Enterprise Linux aus.

Die erledigte Aufgabe entspricht:

```
tasks:
  #Fail Fast Message
  - name: Show Failed System Requirements Message
    fail:
      msg: "The {{ inventory_hostname }} did not meet minimum reqs."
    when: >
      ansible_memtotal_mb < min_ram_mb or
      ansible_distribution != "RedHat"
```

3. Fügen Sie dem Playbook unter dem Kommentar `#Install all Packages` eine einzelne Aufgabe hinzu, um die neueste Version fehlender Pakete zu installieren. Erforderliche Pakete werden durch die Variable `packages` angegeben, die in der Datei `vars.yml` definiert ist.  
Der Aufgabennamen sollte `Ensure required packages are present` lauten.  
Die erledigte Aufgabe entspricht:

```
#Install all Packages
- name: Ensure required packages are present
  yum:
    name: "{{ packages }}"
    state: latest
```

4. Fügen Sie dem Playbook unter dem Kommentar `#Enable and start services` eine einzelne Aufgabe hinzu, um alle Services zu starten. Alle von der Variablen `services` angegebenen Services, die in der Datei `vars.yml` definiert sind, sollten gestartet und aktiviert werden. Geben Sie einen entsprechenden Namen für die Aufgabe an.

Die erledigte Aufgabe entspricht:

```
#Enable and start services
- name: Ensure services are started and enabled
  service:
    name: "{{ item }}"
    state: started
    enabled: yes
  loop: "{{ services }}"
```

5. Fügen Sie unter dem Kommentar `#Block of config tasks` dem Playbook einen Aufgabenblock hinzu. Dieser Block enthält zwei Aufgaben:

- Eine Aufgabe, um sicherzustellen, dass das in der Variable `ssl_cert_dir` angegebene Verzeichnis auf dem Remote-Host vorhanden ist. In diesem Verzeichnis werden die Zertifikate des Webservers gespeichert.
- Eine Aufgabe, um alle von der Variable `web_config_files` angegebenen Dateien auf den Remote-Host zu kopieren. Überprüfen Sie die Struktur der Variablen `web_config_files` in der Datei `vars.yml`. Konfigurieren Sie die Aufgabe, um jede Datei in das richtige Zielverzeichnis auf dem Remote-Host zu kopieren.

Diese Aufgabe sollte den Handler `restart web service` auslösen, wenn eine dieser Dateien auf dem Remote-Server geändert wird.

Darüber hinaus wird eine Debug-Aufgabe ausgeführt, wenn eine der beiden obigen Aufgaben fehlschlägt. In diesem Fall gibt die Aufgabe die folgende Meldung aus: `One or`

more of the configuration changes failed, but the web service is still active.

Geben Sie einen entsprechenden Namen für alle Aufgaben an.

Der abgeschlossene Aufgabenblock entspricht:

```
#Block of config tasks
- name: Setting up the SSL cert directory and config files
  block:
    - name: Create SSL cert directory
      file:
        path: "{{ ssl_cert_dir }}"
        state: directory

    - name: Copy Config Files
      copy:
        src: "{{ item.src }}"
        dest: "{{ item.dest }}"
      loop: "{{ web_config_files }}"
      notify: restart web service

  rescue:
    - name: Configuration Error Message
      debug:
        msg: >
          One or more of the configuration
          changes failed, but the web service
          is still active.
```

6. Das Playbook konfiguriert den Remote-Host zum Überwachen von HTTPS-Standardanforderungen. Fügen Sie unter dem Kommentar `#Configure the firewall` dem Playbook eine einzelne Aufgabe hinzu, um `firewalld` zu konfigurieren.

Diese Aufgabe sollte sicherstellen, dass der Remote-Host HTTP- und HTTPS-Standardverbindungen zulässt. Diese Konfigurationsänderungen sollten sofort wirksam sein und nach einem Neustart des Systems bestehen bleiben. Geben Sie einen entsprechenden Namen für die Aufgabe an.

Die erledigte Aufgabe entspricht:

```
#Configure the firewall
- name: ensure web server ports are open
  firewalld:
    service: "{{ item }}"
    immediate: true
    permanent: true
    state: enabled
  loop:
    - http
    - https
```

7. Definieren Sie den Handler `restart web service`.

Wenn diese Aufgabe ausgelöst wird, sollte sie den Webservice neu starten, der von der Variablen `web_service` definiert wird, die in der Datei `vars.yml` definiert ist.

Am Ende des Playbooks wird der Abschnitt `handlers` hinzugefügt:

```
handlers:
  - name: restart web service
    service:
      name: "{{ web_service }}"
      state: restarted
```

Das vollständige Playbook enthält Folgendes:

```
---
- name: Playbook Control Lab
  hosts: webservers
  vars_files: vars.yml
  tasks:
    #Fail Fast Message
    - name: Show Failed System Requirements Message
      fail:
        msg: "The {{ inventory_hostname }} did not meet minimum reqs."
        when: >
          ansible_memtotal_mb < min_ram_mb or
          ansible_distribution != "RedHat"

    #Install all Packages
    - name: Ensure required packages are present
      yum:
        name: "{{ packages }}"
        state: latest

    #Enable and start services
    - name: Ensure services are started and enabled
      service:
        name: "{{ item }}"
        state: started
        enabled: yes
      loop: "{{ services }}"

    #Block of config tasks
    - name: Setting up the SSL cert directory and config files
      block:
        - name: Create SSL cert directory
          file:
            path: "{{ ssl_cert_dir }}"
            state: directory

        - name: Copy Config Files
          copy:
            src: "{{ item.src }}"
            dest: "{{ item.dest }}"
          loop: "{{ web_config_files }}"
          notify: restart web service
```

```
rescue:
  - name: Configuration Error Message
    debug:
      msg: >
        One or more of the configuration
        changes failed, but the web service
        is still active.

#Configure the firewall
- name: ensure web server ports are open
  firewalld:
    service: "{{ item }}"
    immediate: true
    permanent: true
    state: enabled
  loop:
    - http
    - https

#Add handlers
handlers:
  - name: restart web service
    service:
      name: "{{ web_service }}"
      state: restarted
```

8. Führen Sie im Projektverzeichnis ~/control-review das Playbook `playbook.yml` aus. Das Playbook sollte fehlerfrei ausgeführt werden und die Ausführung der Handler-Aufgabe auslösen.

```
[student@workstation control-review]$ ansible-playbook playbook.yml

PLAY [Playbook Control Lab] ****
TASK [Gathering Facts] ****
ok: [serverb.lab.example.com]

TASK [Show Failed System Requirements Message] ****
skipping: [serverb.lab.example.com]

TASK [Ensure required packages are present] ****
changed: [serverb.lab.example.com]

TASK [Ensure services are started and enabled] ****
changed: [serverb.lab.example.com] => (item=httpd)
ok: [serverb.lab.example.com] => (item=firewalld)

TASK [Create SSL cert directory] ****
changed: [serverb.lab.example.com]

TASK [Copy Config Files] ****
changed: [serverb.lab.example.com] => (item={'src': 'server.key', 'dest': '/etc/
httpd/conf.d/ssl'})
```

**Kapitel 4 |** Implementieren der Aufgabensteuerung

```

changed: [serverb.lab.example.com] => (item={'src': 'server.crt', 'dest': '/etc/
httpd/conf.d/ssl'})
changed: [serverb.lab.example.com] => (item={'src': 'ssl.conf', 'dest': '/etc/
httpd/conf.d'})
changed: [serverb.lab.example.com] => (item={'src': 'index.html', 'dest': '/var/
www/html'})

TASK [ensure web server ports are open] ****
changed: [serverb.lab.example.com] => (item=http)
changed: [serverb.lab.example.com] => (item=https)

RUNNING HANDLER [restart web service] ****
changed: [serverb.lab.example.com]

PLAY RECAP ****
serverb.lab.example.com      : ok=7      changed=6      unreachable=0      failed=0

```

9. Verifizieren Sie, dass der Webserver nun mit einem selbstsignierten benutzerdefinierten Zertifikat auf HTTPS-Anforderungen reagiert, um die Verbindung zu verschlüsseln. Die Webserverantwort sollte der Zeichenfolge **Configured for both HTTP and HTTPS** entsprechen.

```

[student@workstation control-review]$ curl -k -vv https://serverb.lab.example.com
* About to connect() to serverb.lab.example.com port 443 (#0)
*   Trying 172.25.250.11...
* Connected to serverb.lab.example.com (172.25.250.11) port 443 (#0)
* Initializing NSS with certpath: sql:/etc/pki/nssdb
* skipping SSL peer certificate verification
* SSL connection using TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
* Server certificate:
...output omitted...
* start date: Nov 13 15:52:18 2018 GMT
* expire date: Aug 09 15:52:18 2021 GMT
* common name: serverb.lab.example.com
...output omitted...
< Accept-Ranges: bytes
< Content-Length: 36
< Content-Type: text/html; charset=UTF-8
<
Configured for both HTTP and HTTPS.
* Connection #0 to host serverb.lab.example.com left intact

```

## Bewertung

Führen Sie auf `workstation` den Befehl `lab control-review grade` aus, um den Erfolg dieser Übung zu bestätigen. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie das Skript so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab control-review grade
```

## Beenden

Führen Sie den Befehl `lab control-review finish` aus, um nach der Übung eine Bereinigung durchzuführen.

```
[student@workstation ~]$ lab control-review finish
```

Hiermit ist die praktische Übung abgeschlossen.

# Zusammenfassung

---

In diesem Kapitel erhielten Sie folgende Informationen:

- Loops werden verwendet, um eine Reihe von Werten, z. B. eine einfache Liste mit Zeichenfolgen oder eine Liste von Hashes oder Dictionaries zu durchlaufen.
- Bedingungen werden verwendet, um Aufgaben oder Plays nur dann auszuführen, wenn bestimmte Bedingungen erfüllt wurden.
- Handler sind spezielle Aufgaben, die am Ende des Plays ausgeführt werden, wenn sie durch andere Aufgaben benachrichtigt werden.
- Handler werden nur benachrichtigt, wenn eine Aufgabe meldet, dass sich auf einem verwalteten Host etwas geändert hat.
- Aufgaben können so konfiguriert werden, dass sie Fehlerbedingungen behandeln. Dabei können fehlgeschlagene Aufgaben ignoriert werden, Aufrufe von Handlern erzwungen werden, selbst wenn die Aufgabe fehlgeschlagen ist, Aufgaben als fehlgeschlagen markiert werden, selbst wenn sie erfolgreich waren, oder es kann das Verhalten außer Kraft gesetzt werden, das dazu führt, dass eine Aufgabe als geändert kennzeichnet wird.
- Blöcke können verwendet werden, um Aufgaben als eine Einheit zu gruppieren und andere Aufgaben abhängig davon auszuführen, ob alle Aufgaben in dem Block erfolgreich waren.

## Kapitel 5

# Bereitstellen von Dateien auf verwalteten Hosts

### Ziel

Bereitstellen, Verwalten und Anpassen von Dateien auf Hosts, die von Ansible verwaltet werden.

### Ziele

- Dateien auf verwalteten Hosts erstellen, installieren, bearbeiten und entfernen und Berechtigungen, Besitz, SELinux-Kontext und andere Merkmale dieser Dateien verwalten.
- Dateien auf verwalteten Hosts bereitstellen, die mithilfe von Jinja2-Vorlagen angepasst werden.

### Abschnitte

- Ändern und Kopieren von Dateien auf Hosts (und angeleitete Übung)
- Bereitstellen benutzerdefinierter Dateien mit Jinja2-Vorlagen (und angeleitete Übung)

### Praktische Übung

- Bereitstellen von Dateien auf verwalteten Hosts

# Ändern und Kopieren von Dateien auf Hosts

## Ziele

In diesem Abschnitt wird das Erstellen, Installieren, Bearbeiten und Entfernen von Dateien auf verwalteten Hosts sowie das Verwalten von Berechtigungen, der Inhaberschaft, des SELinux-Kontexts und anderer Merkmale dieser Dateien beschrieben.

## Beschreiben von Dateimodulen

Red Hat Ansible Automation Platform enthält standardmäßig eine große Sammlung an Modulen (die „Modul-Library“), die im Rahmen des Ansible-Upstream-Projekts entwickelt werden. Um sie einfacher organisieren, dokumentieren und verwalten zu können, sind sie in der Dokumentation basierend auf der Funktion in Gruppen organisiert, ebenso wie bei der Installation auf einem System.

Die Modul-Library **Files** enthält Module, mit denen Sie die meisten Linux-Dateiverwaltungsaufgaben erledigen können. Dazu zählen beispielsweise das Erstellen, Kopieren, Bearbeiten und Ändern von Berechtigungen und anderer Attribute von Dateien. Die folgende Tabelle enthält eine Liste häufig verwendetener Dateiverwaltungsmodule:

### Häufig verwendete Dateimodule

| Modulname   | Beschreibung des Moduls                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| blockinfile | Fügen Sie einen mehrzeiligen Textblock, der von anpassbaren Markierungslinien umgeben ist, ein, aktualisieren oder entfernen Sie ihn.                                                                                                                                                                                                                                                                                                                      |
| copy        | Kopieren Sie eine Datei vom lokalen oder Remote-Computer an einen Speicherort auf einem verwalteten Host. Analog zum Modul <b>file</b> kann das Modul <b>copy</b> auch Dateiattribute, einschließlich des SELinux-Kontexts, festlegen.                                                                                                                                                                                                                     |
| fetch       | Dieses Modul funktioniert wie das Modul <b>copy</b> , jedoch umgekehrt. Dieses Modul wird verwendet, um auf dem Steuerknoten Dateien von Remote-Rechnern abzurufen und sie in einer Dateistruktur, organisiert nach dem Hostnamen zu speichern.                                                                                                                                                                                                            |
| file        | Legen Sie Attribute fest, beispielsweise Berechtigungen, die Inhaberschaft, SELinux-Kontexte und Zeitstempel von regulären Dateien, Symlinks, Hardlinks und Verzeichnisse. Dieses Modul kann auch reguläre Dateien, Symlinks, Hardlinks und Verzeichnisse erstellen oder entfernen. Eine Reihe anderer dateibezogener Module unterstützen dieselben Optionen, um Attribute als das Modul <b>file</b> festzulegen. Dazu gehört auch das Modul <b>copy</b> . |

| Modulname   | Beschreibung des Moduls                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| lineinfile  | Stellen Sie sicher, dass sich eine bestimmte Zeile in einer Datei befindet, oder ersetzen Sie eine vorhandene Zeile mit einem regulären Rückverweis. Dieses Modul ist vor allem nützlich, wenn Sie eine einzelne Zeile in einer Datei ändern möchten.                                                                                                                                                                                           |
| stat        | Rufen Sie ähnlich wie mit dem Linux-Befehl stat Statusinformationen für eine Datei ab.                                                                                                                                                                                                                                                                                                                                                          |
| synchronize | Ein Wrapper um den Befehl rsync, damit allgemeine Aufgaben schnell und einfach von der Hand gehen. Das Modul synchronize ist nicht dazu vorgesehen, den Zugriff auf die volle Leistung des Befehls rsync bereitzustellen, doch lassen sich dadurch die meisten allgemeinen Aufrufe einfacher implementieren. Möglicherweise müssen Sie entsprechend Ihrem Anwendungsfall den Befehl rsync weiterhin direkt über das Modul run command aufrufen. |

## Automatisierungsbeispiele mit Dateimodulen

Das Erstellen, Kopieren, Bearbeiten und Entfernen von Dateien auf verwalteten Hosts sind allgemeine Aufgaben, die Sie mit Modulen aus der Modul-Library Files implementieren können. Die folgenden Beispiele zeigen Möglichkeiten, wie Sie diese Module verwenden können, um allgemeine Dateiverwaltungsaufgaben zu automatisieren.

### Sicherstellen, dass eine Datei auf verwalteten Hosts vorhanden ist

Verwenden Sie das Modul file, um eine Datei auf verwalteten Hosts abzurufen. Dies funktioniert wie der Befehl touch, wobei eine leere Datei erstellt wird, falls keine vorhanden ist, und die zugehörige Änderungszeit aktualisiert wird, sofern sie vorhanden ist. In diesem Beispiel stellt Ansible zusätzlich zum Abrufen der Datei sicher, dass der Benutzer, dem die Datei gehört, die Gruppe und die Berechtigungen der Datei auf bestimmte Werte festgelegt sind.

```
- name: Touch a file and set permissions
  file:
    path: /path/to/file
    owner: user1
    group: group1
    mode: 0640
    state: touch
```

Beispielergebnis:

```
[user@host ~]$ ls -l file
-rw-r-----. user1 group1 0 Nov 25 08:00 file
```

### Ändern von Dateiattributen

Sie können das Modul file verwenden, um sicherzustellen, dass eine neue oder vorhandene Datei sowohl die richtigen Berechtigungen als auch den SELinux-Typ aufweist.

## Kapitel 5 | Bereitstellen von Dateien auf verwalteten Hosts

Beispielsweise hat die folgende Datei den SELinux-Standardkontext relativ zum Benutzerverzeichnis eines Benutzers beibehalten, der nicht dem gewünschten Kontext entspricht.

```
[user@host ~]$ ls -Z samba_file  
-rw-r--r--. owner group unconfined_u:object_r:user_home_t:s0 samba_file
```

Die folgende Aufgabe stellt sicher, dass das SELinux-Kontexttypattribut der Datei `samba_file` der gewünschte Typ `samba_share_t` ist. Dieses Verhalten ähnelt dem Linux-Befehl `chcon`.

```
- name: SELinux type is set to samba_share_t  
  file:  
    path: /path/to/samba_file  
    setype: samba_share_t
```

Beispielergebnis:

```
[user@host ~]$ ls -Z samba_file  
-rw-r--r--. owner group unconfined_u:object_r:samba_share_t:s0 samba_file
```



### Anmerkung

Dateiattributparameter sind in mehreren Dateiverwaltungsmodulen verfügbar. Führen Sie die Befehle `ansible-doc file` und `ansible-doc copy` aus, um zusätzliche Informationen abzurufen.

## Vornehmen dauerhafter Änderungen des SELinux-Dateikontexts

Das Modul `file` fungiert wie `chcon` beim Festlegen von Dateikontexten. Mit diesem Modul vorgenommene Änderungen könnten durch Ausführen von `restorecon` unerwartet rückgängig gemacht werden. Nachdem `file` zum Festlegen des Kontexts verwendet wurde, können Sie `sefcontext` aus der Sammlung der System-Module verwenden, um die SELinux-Richtlinie wie `semanage fcontext` zu aktualisieren.

```
- name: SELinux type is persistently set to samba_share_t  
  sefcontext:  
    target: /path/to/samba_file  
    setype: samba_share_t  
    state: present
```

Beispielergebnis:

```
[user@host ~]$ ls -Z samba_file  
-rw-r--r--. owner group unconfined_u:object_r:samba_share_t:s0 samba_file
```



### Wichtig

Das Modul `sefcontext` aktualisiert den Standardkontext für das Ziel in der SELinux-Richtlinie, ändert jedoch nicht den Kontext für vorhandene Dateien.

## Kopieren und Bearbeiten von Dateien auf verwalteten Hosts

In diesem Beispiel wird das Modul `copy` verwendet, um eine im Ansible-Arbeitsverzeichnis auf dem Kontrollknoten befindliche Datei auf ausgewählte verwaltete Knoten zu kopieren.

Dieses Modul geht standardmäßig davon aus, dass `force: yes` festgelegt ist. Dadurch wird das Modul gezwungen, die Remote-Datei zu überschreiben, sofern sie vorhanden ist, jedoch andere Inhalte aus der zu kopierenden Datei enthält. Wenn `force: no` festgelegt ist, wird nur die Datei auf den verwalteten Host kopiert, sofern sie noch nicht vorhanden ist.

```
- name: Copy a file to managed hosts
  copy:
    src: file
    dest: /path/to/file
```

Verwenden Sie zum Abrufen von Dateien von verwalteten Hosts das `fetch`-Modul. Dies kann verwendet werden, um eine Datei wie einen öffentlichen SSH-Schlüssel von einem Referenzsystem abzurufen, bevor sie an andere verwaltete Hosts verteilt wird.

```
- name: Retrieve SSH key from reference host
  fetch:
    src: "/home/{{ user }}/.ssh/id_rsa.pub"
    dest: "files/keys/{{ user }}.pub"
```

Verwenden Sie das Modul `lineinfile`, um sicherzustellen, dass eine spezifische Textzeile in einer vorhandenen Datei vorhanden ist:

```
- name: Add a line of text to a file
  lineinfile:
    path: /path/to/file
    line: 'Add this line to the file'
    state: present
```

Verwenden Sie das Modul `blockinfile`, um einer vorhandenen Datei einen Textblock hinzuzufügen:

```
- name: Add additional lines to a file
  blockinfile:
    path: /path/to/file
    block: |
      First line in the additional block of text
      Second line in the additional block of text
    state: present
```



### Anmerkung

Bei der Verwendung des Moduls `blockinfile` werden zum Gewährleisten der Idempotenz kommentierte Blockmarker am Anfang und Ende des Blocks eingefügt.

```
# BEGIN ANSIBLE MANAGED BLOCK
First line in the additional block of text
Second line in the additional block of text
# END ANSIBLE MANAGED BLOCK
```

Sie können den Parameter `marker` für das Modul verwenden, um sicherzustellen, dass das richtige Kommentarzeichen oder der richtige Text für die betreffende Datei verwendet wird.

## Entfernen einer Datei von verwalteten Hosts

Ein grundlegendes Beispiel für das Entfernen einer Datei von verwalteten Hosts besteht in der Verwendung des Moduls `file` mit dem Parameter `state: absent`. Der Parameter `state` ist für viele Module optional. Aus mehreren Gründen sollten Sie Ihre Absichten immer deutlich machen, unabhängig davon, ob Sie `state: present` oder `state: absent` verwenden möchten. Einige Module unterstützen auch andere Optionen. Es ist möglich, dass sich der Standardwert zu einem bestimmten Zeitpunkt ändern kann. Am wichtigsten ist es jedoch möglicherweise basierend auf Ihrer Aufgabe zu verstehen, in welchem Zustand sich das System befinden sollte.

```
- name: Make sure a file does not exist on managed hosts
  file:
    dest: /path/to/file
    state: absent
```

## Abrufen des Status einer Datei auf verwalteten Hosts

Das Modul `stat` ruft ähnlich wie der Linux-Befehl `stat` Fakten für eine Datei ab. Mit Parametern können u. a. Dateiattribute abgerufen und die Prüfsumme einer Datei bestimmt werden.

Das Modul `stat` gibt ein Hash-Wörterbuch der Werte zurück, welche die Dateistatusdaten enthalten. Dadurch können Sie sich mit separaten Variablen auf einzelne Informationen beziehen.

Im folgenden Beispiel werden die Ergebnisse des Moduls `stat` registriert. Anschließend wird die MD5-Prüfsumme der überprüften Datei ausgegeben. (Der moderne SHA256-Algorithmus ist ebenfalls verfügbar. MD5 wird hier verwendet, damit die Beispielausgabe lesbarer wird.)

```
- name: Verify the checksum of a file
  stat:
    path: /path/to/file
    checksum_algorithm: md5
  register: result

- debug
  msg: "The checksum of the file is {{ result.stat.checksum }}"
```

Die Ausgabe sollte in etwa wie folgt aussehen:

```
TASK [Get md5 checksum of a file] *****
ok: [hostname]

TASK [debug] *****
ok: [hostname] => {
    "msg": "The checksum of the file is 5f76590425303022e933c43a7f2092a3"
}
```

Informationen zu den vom Modul `stat` zurückgegebenen Werten sind durch `ansible-doc` dokumentiert. Alternativ können Sie eine Variable registrieren und ihre Inhalte anzeigen, um zu sehen, was verfügbar ist:

```
- name: Examine all stat output of /etc/passwd
hosts: localhost

tasks:
  - name: stat /etc/passwd
    stat:
      path: /etc/passwd
    register: results

  - name: Display stat results
    debug:
      var: results
```

## Synchronisieren von Dateien zwischen dem Kontrollknoten und verwalteten Hosts

Das Modul `synchronize` fungiert als Wrapper um das Tool `rsync`, wodurch sich allgemeine Dateiverwaltungsaufgaben in Ihren Playbooks vereinfachen lassen. Das Tool `rsync` muss auf dem lokalen und Remote-Host installiert sein. Bei Verwendung des Moduls `synchronize` ist der „lokale Host“ der Host, von dem die Synchronisierungsaufgabe stammt (in der Regel der Kontrollknoten). Demgegenüber ist der „Zielhost“ der Host, mit dem `synchronize` eine Verbindung herstellt.

Im folgenden Beispiel wird eine im Ansible-Arbeitsverzeichnis befindliche Datei mit den verwalteten Hosts synchronisiert:

```
- name: synchronize local file to remote files
  synchronize:
    src: file
    dest: /path/to/file
```

Es gibt viele Möglichkeiten, das Modul `synchronize` und die vielen zugehörigen Parameter zu verwenden. Dazu zählt auch das Synchronisieren von Verzeichnissen. Führen Sie den Befehl `ansible-doc synchronize` aus, um zusätzliche Parameter und Playbook-Beispiele abzurufen.



### Literaturhinweise

Manpages `ansible-doc(1)`, `chmod(1)`, `chown(1)`, `rsync(1)`, `stat(1)` und `touch(1)`

### Dateimodule

[https://docs.ansible.com/ansible/2.9/modules/list\\_of\\_files\\_modules.html](https://docs.ansible.com/ansible/2.9/modules/list_of_files_modules.html)

## ► Angeleitete Übung

# Ändern und Kopieren von Dateien auf Hosts

In dieser Übung verwenden Sie Ansible-Standardmodule, um Dateien auf verwalteten Hosts zu erstellen, zu installieren, zu bearbeiten und zu entfernen und um die Berechtigungen, die Inhaberschaft und die SELinux-Kontexte dieser Dateien zu verwalten.

## Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Rufen Sie Dateien von den verwalteten Hosts nach Hostname ab und speichern Sie diese lokal.
- Erstellen Sie Playbooks, die allgemeine Dateiverwaltungsmodule verwenden, beispielsweise `copy`, `file`, `lineinfile` und `blockinfile`.

## Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab file-manage start` aus. Das Skript erstellt das Projektverzeichnis `file-manage` und lädt die Ansible-Konfigurationsdatei und die Hostinventardatei herunter, die für die Übung erforderlich sind.

```
[student@workstation ~]$ lab file-manage start
```

## Anweisungen

- 1. Wechseln Sie als Benutzer `student` auf `workstation` in das Arbeitsverzeichnis `/home/student/file-manage`. Erstellen Sie ein Playbook mit dem Namen `secure_log_backups.yml` im aktuellen Arbeitsverzeichnis. Konfigurieren Sie das Playbook so, dass das Modul `fetch` verwendet wird, um die Protokolldatei `/var/log/secure` von jedem der verwalteten Hosts abzurufen und sie auf dem Kontrollknoten zu speichern. Das Playbook sollte das Verzeichnis `secure-backups` mit nach dem Hostnamen benannten Unterverzeichnissen auf jedem verwalteten Host erstellen. Speichern Sie die Backup-Dateien in den entsprechenden Unterverzeichnissen.

- 1.1. Wechseln Sie in das Arbeitsverzeichnis `/home/student/file-manage`.

```
[student@workstation ~]$ cd ~/file-manage  
[student@workstation file-manage]$
```

- 1.2. Erstellen Sie das Playbook `secure_log_backups.yml` mit dem folgenden anfänglichen Inhalt:

```
---
```

```
- name: Use the fetch module to retrieve secure log files
  hosts: all
  remote_user: root
```

- 1.3. Fügen Sie dem Playbook `secure_log_backups.yml` eine Aufgabe hinzu, welche die Protokolldatei `/var/log/secure` von den verwalteten Hosts abruft und sie im Verzeichnis `~/file-manage/secure-backups` speichert. Das Modul `fetch` erstellt das Verzeichnis `~/file-manage/secure-backups`, falls es noch nicht vorhanden ist. Verwenden Sie den Parameter `flat: no`, um sicherzustellen, dass dem Ziel standardmäßig der Hostname, Pfad und Dateiname angehängt werden:

```
tasks:
- name: Fetch the /var/log/secure log file from managed hosts
  fetch:
    src: /var/log/secure
    dest: secure-backups
    flat: no
```

- 1.4. Führen Sie vor dem Ausführen des Playbooks den Befehl `ansible-playbook --syntax-check secure_log_backups.yml` aus, um die zugehörige Syntax zu verifizieren. Korrigieren Sie sämtliche Fehler, bevor Sie mit dem nächsten Schritt fortfahren.

```
[student@workstation file-manage]$ ansible-playbook --syntax-check \
> secure_log_backups.yml

playbook: secure_log_backups.yml
```

- 1.5. Führen Sie `ansible-playbook secure_log_backups.yml` aus, um das Playbook auszuführen:

```
[student@workstation file-manage]$ ansible-playbook secure_log_backups.yml
PLAY [Use the fetch module to retrieve secure log files] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]
ok: [serverb.lab.example.com]

TASK [Fetch the /var/log/secure file from managed hosts] ****
changed: [serverb.lab.example.com]
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com      : ok=2    changed=1    unreachable=0    failed=0
serverb.lab.example.com      : ok=2    changed=1    unreachable=0    failed=0
```

- 1.6. Verifizieren Sie die Playbook-Ergebnisse:

```
[student@workstation file-manage]$ tree -F secure-backups
secure-backups
└── servera.lab.example.com/
    └── var/
        └── log/
            └── secure
└── serverb.lab.example.com/
    └── var/
        └── log/
            └── secure
```

- 2. Erstellen Sie das Playbook `copy_file.yml` im aktuellen Arbeitsverzeichnis. Konfigurieren Sie das Playbook so, dass die Datei `/home/student/file-manage/files/users.txt` als Root-Benutzer auf alle verwalteten Hosts kopiert wird.

- 2.1. Fügen Sie dem Playbook `copy_file.yml` den folgenden anfänglichen Inhalt hinzu:

```
---
- name: Using the copy module
  hosts: all
  remote_user: root
```

- 2.2. Fügen Sie dem Modul `copy` eine Aufgabe hinzu, um die Datei `/home/student/file-manage/files/users.txt` auf alle verwalteten Hosts zu kopieren. Verwenden Sie das Modul `copy`, um die folgenden Parameter für die Datei `users.txt` festzulegen:

| Parameter | Werte                  |
|-----------|------------------------|
| src       | files/users.txt        |
| dest      | /home/devops/users.txt |
| owner     | devops                 |
| group     | devops                 |
| Modus     | u+rwx, g-wx, o-rwx     |
| setype    | samba_share_t          |

```
tasks:
  - name: Copy a file to managed hosts and set attributes
    copy:
      src: files/users.txt
      dest: /home/devops/users.txt
      owner: devops
      group: devops
      mode: u+rwx, g-wx, o-rwx
      setype: samba_share_t
```

**Kapitel 5 |** Bereitstellen von Dateien auf verwalteten Hosts

- 2.3. Führen Sie den Befehl `ansible-playbook --syntax-check copy_file.yml` aus, um die Syntax des Playbooks `copy_file.yml` zu überprüfen.

```
[student@workstation file-manage]$ ansible-playbook --syntax-check copy_file.yml  
playbook: copy_file.yml
```

- 2.4. Führen Sie das Playbook aus:

```
[student@workstation file-manage]$ ansible-playbook copy_file.yml  
PLAY [Using the copy module] *****  
  
TASK [Gathering Facts] *****  
ok: [serverb.lab.example.com]  
ok: [servera.lab.example.com]  
  
TASK [Copy a file to managed hosts and set attributes] *****  
changed: [servera.lab.example.com]  
changed: [serverb.lab.example.com]  
  
PLAY RECAP *****  
servera.lab.example.com : ok=2     changed=1      unreachable=0      failed=0  
serverb.lab.example.com : ok=2     changed=1      unreachable=0      failed=0
```

- 2.5. Führen Sie einen Ad-hoc-Befehl aus, um den Befehl `ls -Z` als der Benutzer `devops` auszuführen, um die Attribute der Datei `users.txt` auf den verwalteten Hosts zu verifizieren.

```
[student@workstation file-manage]$ ansible all -m command -a 'ls -Z' -u devops  
servera.lab.example.com | CHANGED | rc=0 >>  
unconfined_u:object_r:samba_share_t:s0 users.txt  
  
serverb.lab.example.com | CHANGED | rc=0 >>  
unconfined_u:object_r:samba_share_t:s0 users.txt
```

- 3. In einem vorherigen Schritt wurde das Feld `samba_share_t` des SELinux-Typs für die Datei `users.txt` festgelegt. Nun wird jedoch bestimmt, dass die Standardwerte für den SELinux-Dateikontext festgelegt werden sollten.  
Erstellen Sie ein Playbook mit dem Namen `selinux_defaults.yml` im aktuellen Arbeitsverzeichnis. Konfigurieren Sie das Playbook für die Verwendung des Moduls `file`, um den SELinux-Standardkontext für die Felder `user`, `role`, `type` und `level` sicherzustellen.

**Anmerkung**

In der realen Welt würden Sie `copy_file.yml` ebenfalls bearbeiten und das Keyword `setype` entfernen.

- 3.1. Erstellen Sie das Playbook `selinux_defaults.yml`:

```
---
- name: Using the file module to ensure SELinux file context
  hosts: all
  remote_user: root
  tasks:
    - name: SELinux file context is set to defaults
      file:
        path: /home/devops/users.txt
        seuser: _default
        serole: _default
        setype: _default
        selevel: _default
```

- 3.2. Führen Sie den Befehl `ansible-playbook --syntax-check selinux_defaults.yml` aus, um die Syntax des Playbooks `selinux_defaults.yml` zu überprüfen.

```
[student@workstation file-manage]$ ansible-playbook --syntax-check \
> selinux_defaults.yml

playbook: selinux_defaults.yml
```

- 3.3. Führen Sie das Playbook aus:

```
[student@workstation file-manage]$ ansible-playbook selinux_defaults.yml
PLAY [Using the file module to ensure SELinux file context] ****
TASK [Gathering Facts] ****
ok: [serverb.lab.example.com]
ok: [servera.lab.example.com]

TASK [SELinux file context is set to defaults] ****
changed: [serverb.lab.example.com]
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=2     changed=1      unreachable=0      failed=0
serverb.lab.example.com : ok=2     changed=1      unreachable=0      failed=0
```

- 3.4. Führen Sie einen Ad-hoc-Befehl aus, um den Befehl `ls -Z` als der Benutzer `devops` auszuführen, um die Standarddateiattribute von `unconfined_u:object_r:user_home_t:s0` zu verifizieren.

```
[student@workstation file-manage]$ ansible all -m command -a 'ls -Z' -u devops
servera.lab.example.com | CHANGED | rc=0 >>
unconfined_u:object_r:user_home_t:s0 users.txt

serverb.lab.example.com | CHANGED | rc=0 >>
unconfined_u:object_r:user_home_t:s0 users.txt
```

**Kapitel 5 |** Bereitstellen von Dateien auf verwalteten Hosts

- 4. Erstellen Sie ein Playbook mit dem Namen `add_line.yml` im aktuellen Arbeitsverzeichnis. Konfigurieren Sie das Playbook so, dass das Modul `lineinfile` die Zeile `This line was added by the lineinfile module.` an die Datei `/home/devops/users.txt` auf allen verwalteten Hosts anhängt.

4.1. Erstellen Sie das Playbook `add_line.yml`:

```
---
- name: Add text to an existing file
  hosts: all
  remote_user: devops
  tasks:
    - name: Add a single line of text to a file
      lineinfile:
        path: /home/devops/users.txt
        line: This line was added by the lineinfile module.
        state: present
```

4.2. Führen Sie den Befehl `ansible-playbook --syntax-check add_line.yml` aus, um die Syntax des Playbooks `add_line.yml` zu überprüfen.

```
[student@workstation file-manage]$ ansible-playbook --syntax-check add_line.yml
playbook: add_line.yml
```

4.3. Führen Sie das Playbook aus:

```
[student@workstation file-manage]$ ansible-playbook add_line.yml
PLAY [Add text to an existing file] ****
TASK [Gathering Facts] ****
ok: [serverb.lab.example.com]
ok: [servera.lab.example.com]

TASK [Add a single line of text to a file] ****
changed: [servera.lab.example.com]
changed: [serverb.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
serverb.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```

4.4. Verwenden Sie das Modul `command` mit der Option `cat` als Benutzer `devops`, um den Inhalt der Datei `users.txt` auf den verwalteten Hosts zu verifizieren.

```
[student@workstation file-manage]$ ansible all -m command \
> -a 'cat users.txt' -u devops
serverb.lab.example.com | CHANGED | rc=0 >>
This line was added by the lineinfile module.

servera.lab.example.com | CHANGED | rc=0 >>
This line was added by the lineinfile module.
```

- 5. Erstellen Sie ein Playbook mit dem Namen `add_block.yml` im aktuellen Arbeitsverzeichnis. Konfigurieren Sie das Playbook so, dass das Modul `blockinfile` den folgenden Textblock auf allen verwalteten Hosts an die Datei `/home/devops/users.txt` anhängt.

```
This block of text consists of two lines.  
They have been added by the blockinfile module.
```

- 5.1. Erstellen Sie das Playbook `add_block.yml`:

```
---  
- name: Add block of text to a file  
  hosts: all  
  remote_user: devops  
  tasks:  
    - name: Add a block of text to an existing file  
      blockinfile:  
        path: /home/devops/users.txt  
        block: |  
          This block of text consists of two lines.  
          They have been added by the blockinfile module.  
        state: present
```

- 5.2. Führen Sie den Befehl `ansible-playbook --syntax-check add_block.yml` aus, um die Syntax des Playbooks `add_block.yml` zu überprüfen.

```
[student@workstation file-manage]$ ansible-playbook --syntax-check add_block.yml  
playbook: add_block.yml
```

- 5.3. Führen Sie das Playbook aus:

```
[student@workstation file-manage]$ ansible-playbook add_block.yml  
  
PLAY [Add block of text to a file] *****  
  
TASK [Gathering Facts] *****  
ok: [serverb.lab.example.com]  
ok: [servera.lab.example.com]  
  
TASK [Add a block of text to an existing file] *****  
changed: [servera.lab.example.com]  
changed: [serverb.lab.example.com]  
  
PLAY RECAP *****  
servera.lab.example.com : ok=2     changed=1     unreachable=0    failed=0  
serverb.lab.example.com : ok=2     changed=1     unreachable=0    failed=0
```

- 5.4. Verwenden Sie das Modul `command` mit dem Befehl `cat`, um den richtigen Inhalt der Datei `/home/devops/users.txt` auf dem verwalteten Host zu verifizieren.

```
[student@workstation file-manage]$ ansible all -m command \
> -a 'cat users.txt' -u devops
serverb.lab.example.com | CHANGED | rc=0 >>
This line was added by the lineinfile module.
# BEGIN ANSIBLE MANAGED BLOCK
This block of text consists of two lines.
They have been added by the blockinfile module.
# END ANSIBLE MANAGED BLOCK

servera.lab.example.com | CHANGED | rc=0 >>
This line was added by the lineinfile module.
# BEGIN ANSIBLE MANAGED BLOCK
This block of text consists of two lines.
They have been added by the blockinfile module.
# END ANSIBLE MANAGED BLOCK
```

- 6. Erstellen Sie ein Playbook mit dem Namen `remove_file.yml` im aktuellen Arbeitsverzeichnis. Konfigurieren Sie das Playbook so, dass das Modul `file` verwendet wird, um die Datei `/home/devops/users.txt` von allen verwalteten Hosts zu entfernen.

- 6.1. Erstellen Sie das Playbook `remove_file.yml`:

```
---
- name: Use the file module to remove a file
  hosts: all
  remote_user: devops
  tasks:
    - name: Remove a file from managed hosts
      file:
        path: /home/devops/users.txt
        state: absent
```

- 6.2. Führen Sie den Befehl `ansible-playbook --syntax-check remove_file.yml` aus, um die Syntax des Playbooks `remove_file.yml` zu überprüfen.

```
[student@workstation file-manage]$ ansible-playbook --syntax-check remove_file.yml
playbook: remove_file.yml
```

- 6.3. Führen Sie das Playbook aus:

```
[student@workstation file-manage]$ ansible-playbook remove_file.yml
PLAY [Use the file module to remove a file] ****
TASK [Gathering Facts] ****
ok: [serverb.lab.example.com]
ok: [servera.lab.example.com]

TASK [Remove a file from managed hosts] ****
changed: [serverb.lab.example.com]
changed: [servera.lab.example.com]
```

```
PLAY RECAP ****
servera.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
serverb.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```

- 6.4. Führen Sie einen Ad-hoc-Befehl aus, um den Befehl `ls -l` auszuführen, damit bestätigt wird, dass die Datei `users.txt` nicht mehr auf den verwalteten Hosts vorhanden ist.

```
[student@workstation file-manage]$ ansible all -m command -a 'ls -l' -u devops
serverb.lab.example.com | CHANGED | rc=0 >>
total 0

servera.lab.example.com | CHANGED | rc=0 >>
total 0
```

## Beenden

Führen Sie auf `workstation` das Skript `lab file-manage finish` aus, um diese Übung zu bereinigen.

```
[student@workstation ~]$ lab file-manage finish
```

Hiermit ist die angeleitete Übung beendet.

# Bereitstellen benutzerdefinierter Dateien mit Jinja2-Vorlagen

---

## Ziele

Am Ende dieses Abschnitts sollten Sie in der Lage sein, Dateien auf verwalteten Hosts bereitzustellen, die mithilfe von Jinja2-Vorlagen angepasst werden.

## Erstellen von Vorlagen aus Dateien

Red Hat Ansible Automation Platform verfügt über eine Reihe von Modulen, mit denen vorhandene Dateien geändert werden können. Dazu zählen u. a. `lineinfile` und `blockinfile`. Sie sind jedoch nicht immer einfach und ordnungsgemäß zu verwenden.

Eine wesentlich leistungsfähigere Methode zum Verwalten von Dateien besteht darin, sie in *Vorlagen umzuwandeln*. Mit dieser Methode können Sie mithilfe von Ansible-Variablen und -Fakten eine Vorlagenkonfigurationsdatei schreiben, die automatisch für den verwalteten Host angepasst wird, wenn die Datei bereitgestellt wird. Dies kann einfacher zu kontrollieren sein und ist weniger fehleranfällig.

## Einführung in Jinja2

Ansible verwendet das Templating-System Jinja2 für Vorlagendateien. Ansible verwendet auch die Jinja2-Syntax, um auf Variablen in Playbooks zu verweisen. Sie wissen also bereits ein wenig darüber, wie sie verwendet werden.

## Verwenden von Trennzeichen

Variablen und logische Ausdrücke werden zwischen Tags oder Trennzeichen platziert. Beispiel: Jinja2-Vorlagen verwenden `{% EXPR %}` für Ausdrücke oder logische Operationen (z. B. Loops), wohingegen `{{ EXPR }}` für die Ausgabe der Ergebnisse eines Ausdrucks oder einer Variablen an den Endbenutzer verwendet wird. Das letztgenannte Tag wird beim Rendern durch einen Wert bzw. mehrere Werte ersetzt und kann vom Endbenutzer angezeigt werden. Verwenden Sie die Syntax `{# COMMENT #}`, um Kommentare einzuschließen, die in der finalen Datei nicht angezeigt werden sollen.

Im folgenden Beispiel enthält die erste Zeile einen Kommentar, der nicht zur finalen Datei hinzugefügt wird. Die Variablenreferenzen in der zweiten Zeile werden durch die Werte der Systemfakten ersetzt, auf die verwiesen wird.

```
{# /etc/hosts line #}
{{ ansible_facts['default_ipv4']['address'] }}    {{ ansible_facts['hostname'] }}
```

## Erstellen einer Jinja2-Vorlage

Eine Jinja2-Vorlage besteht aus mehreren Elementen: Daten, Variablen und Ausdrücken. Diese Variablen und Ausdrücke werden durch ihre Werte ersetzt, wenn die Jinja2-Vorlage gerendert ist. Die in der Vorlage genutzten Variablen können im Abschnitt `vars` des Playbooks spezifiziert werden. Es ist möglich, die verwalteten Host-Fakten wie Variablen in einer Vorlage zu verwenden.



### Anmerkung

Denken Sie daran, dass die mit einem verwalteten Host verbundenen Fakten durch den Befehl `ansible system_hostname -i inventory_file -m setup` aufgerufen werden können.

Das folgende Beispiel zeigt, wie Sie eine Vorlage für `/etc/ssh/sshd_config` mit den durch Ansible von verwalteten Host abgerufenen Variablen und Fakten erstellen. Wenn das verbundene Playbook ausgeführt wird, werden diese Fakten durch ihre Werte ersetzt, die auf dem verwalteten Host konfiguriert werden.



### Anmerkung

Eine Datei mit Jinja2-Vorlage benötigt keine spezifische Dateierweiterung (beispielsweise `.j2`). Wenn Sie jedoch eine solche Dateierweiterung bereitstellen, können Sie sich möglicherweise leichter daran erinnern, dass es sich um eine Vorlagendatei handelt.

```
# {{ ansible_managed }}
# DO NOT MAKE LOCAL MODIFICATIONS TO THIS FILE AS THEY WILL BE LOST

Port {{ ssh_port }}
ListenAddress {{ ansible_facts['default_ipv4']['address'] }}

HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_ecdsa_key
HostKey /etc/ssh/ssh_host_ed25519_key

SyslogFacility AUTHPRIV

PermitRootLogin {{ root_allowed }}
AllowGroups {{ groups_allowed }}

AuthorizedKeysFile /etc/.rht_authorized_keys .ssh/authorized_keys

PasswordAuthentication {{ passwords_allowed }}

ChallengeResponseAuthentication no

GSSAPIAuthentication yes
GSSAPICleanupCredentials no

UsePAM yes

X11Forwarding yes
UsePrivilegeSeparation sandbox

AcceptEnv LANG LC_CTYPE LC_NUMERIC LC_TIME LC_COLLATE LC_MONETARY LC_MESSAGES
AcceptEnv LC_PAPER LC_NAME LC_ADDRESS LC_TELEPHONE LC_MEASUREMENT
AcceptEnv LC_IDENTIFICATION LC_ALL LANGUAGE
```

```
AcceptEnv XMODIFIERS

Subsystem sftp /usr/libexec/openssh/sftp-server
```

## Bereitstellen von Jinja2-Vorlagen

Jinja2-Vorlagen sind leistungsstarke Tools für die Bearbeitung von Konfigurationsdateien, die in verwalteten Hosts eingesetzt werden sollen. Wenn eine Jinja2-Vorlage für eine Konfigurationsdatei erstellt ist, kann sie mithilfe des `template`-Moduls auf verwalteten Hosts eingesetzt werden, die den Transfer einer lokalen Datei im Controller-Knoten der verwalteten Hosts unterstützt.

Um das Modul `template` zu verwenden, nutzen Sie die folgende Syntax. Der mit dem Schlüssel `src` verbundene Wert spezifiziert die Jinja2-Quellvorlage, und der mit dem Schlüssel `dest` verbundene Wert spezifiziert die Datei, die auf den Ziel-Hosts erstellt werden soll.

```
tasks:
  - name: template render
    template:
      src: /tmp/j2-template.j2
      dest: /tmp/dest-config-file.txt
```



### Anmerkung

Mit dem Modul `template` können Sie wie beim Modul `file` den Inhaber (den Benutzer, dem die Datei gehört), die Gruppe, die Berechtigungen und den SELinux-Kontext der bereitgestellten Datei angeben. Es kann auch die Option `validate` verwenden, um einen arbiträren Befehl (beispielsweise `visudo -c`) auszuführen, um die Syntax einer Datei auf Richtigkeit zu überprüfen, bevor sie an die entsprechende Stelle kopiert wird.

Weitere Informationen finden Sie in `ansible-doc template`.

## Verwalten von vorlagenbasierten Dateien

Um zu vermeiden, dass Systemadministratoren von Ansible bereitgestellte Dateien ändern, sollten Sie oben in der Vorlage einen Kommentar einfügen, um anzugeben, dass die Datei nicht manuell bearbeitet werden soll.

Dazu können Sie u. a. den Zeichenfolgensatz „Ansible managed“ in der Anweisung `ansible_managed` verwenden. Dies ist keine normale Variable, kann aber als eine in einer Vorlage verwendet werden. Die Anweisung `ansible_managed` wird in der Datei `ansible.cfg` festgelegt:

```
ansible_managed = Ansible managed
```

Um innerhalb einer Jinja2-Vorlage die Zeichenfolge `ansible_managed` einzubinden, nutzen Sie die folgende Syntax:

```
{{ ansible_managed }}
```

## Kontrollstrukturen

Sie können Jinja2-Kontrollstrukturen in Vorlagendateien verwenden, um sich wiederholende Eingaben zu reduzieren, um Eingaben für die jeweiligen Hosts in einem Play dynamisch einzugeben oder um Text bedingt in eine Datei einzufügen.

### Verwenden von Loops

Jinja2 verwendet die Anweisung `for` zur Bereitstellung der Loop-Funktion. Im folgenden Beispiel wird die Variable `user` durch alle Werte ersetzt, die in der Variablen `users` enthalten sind, wobei pro Zeile ein Wert verwendet wird.

```
{% for user in users %}  
    {{ user }}  
{% endfor %}
```

In der folgenden Beispielvorlage wird die Anweisung `for` verwendet, um alle Werte in der Variablen `users` zu durchlaufen, wobei `myuser` durch den jeweiligen Wert ersetzt wird, sofern der Wert nicht `root` lautet.

```
{# for statement #}  
{% for myuser in users if not myuser == "root" %}  
User number {{ loop.index }} - {{ myuser }}  
{% endfor %}
```

Die Variable `loop.index` wird um die Indexnummer erweitert, unter der die Schleife derzeit ausgeführt wird. Bei der ersten Ausführung des Loops lautete der Wert „1“. Dieser Wert wird nach jeder Wiederholung um 1 erhöht.

Weiteres Beispiel: Diese Vorlage verwendet zudem die Anweisung `for` und nimmt an, dass in der verwendeten Inventardatei die Variable `myhosts` definiert wurde. Diese Variable enthält eine Liste von Hosts, die verwaltet werden sollten. Mit der folgenden `for`-Anweisung werden alle Hosts in der Gruppe `myhosts` aus dem Inventar in der Datei gelistet.

```
{% for myhost in groups['myhosts'] %}  
{{ myhost }}  
{% endfor %}
```

Für ein praktischeres Beispiel können Sie dies verwenden, um eine `/etc/hosts`-Datei dynamisch anhand von Hostfakten zu generieren. Angenommen, Sie verfügen über folgendes Playbook:

```
- name: /etc/hosts is up to date  
  hosts: all  
  gather_facts: yes  
  tasks:  
    - name: Deploy /etc/hosts  
      template:  
        src: templates/hosts.j2  
        dest: /etc/hosts
```

Die folgende, dreizeilige `templates/hosts.j2`-Vorlage erstellt die Datei anhand sämtlicher Hosts in der Gruppe `all`. (Die mittlere Zeile ist aufgrund der Länge der Variablennamen äußerst

lang in der Vorlage.) Sie durchläuft jeden Host in der Gruppe, um drei Fakten für die Datei `/etc/hosts` abzurufen.

```
{% for host in groups['all'] %}  
{{ hostvars[host]['ansible_facts']['default_ipv4']['address'] }} {{ hostvars[host]  
['ansible_facts']['fqdn'] }} {{ hostvars[host]['ansible_facts']['hostname'] }}  
{% endfor %}
```

## Verwenden von Bedingungen

Jinja2 verwendet die Anweisung `if`, um Funktionen für die bedingte Steuerung bereitzustellen. Dadurch können Sie eine Zeile in einer bereitgestellten Datei einfügen, sofern bestimmte Bedingungen erfüllt sind.

Im folgenden Beispiel wird der Wert der Variablen `result` nur dann in der bereitgestellten Datei platziert, wenn der Wert der Variablen `finished` dem Wert `True` entspricht.

```
{% if finished %}  
{{ result }}  
{% endif %}
```



### Wichtig

Sie können Jinja2-Loops und -Bedingungen in Ansible-Vorlagen verwenden, nicht aber in Ansible-Playbooks.

## Variablenfilter

Jinja2 bietet Filter, mit denen das Ausgabeformat für Vorlagenausdrücke geändert werden kann (z. B. zu JSON). Es sind u. a. Filter für die Sprachen YAML und JSON verfügbar. Der Filter `to_json` formatiert die Ausgabe des Ausdrucks mithilfe von JSON und der Filter `to_yaml` formatiert die Ausgabe des Ausdrucks mithilfe von YAML.

```
{{ output | to_json }}  
{{ output | to_yaml }}
```

Es sind weitere Filter verfügbar, beispielsweise `to_nice_json` und `to_nice_yaml`. Diese formatieren die Ausgabe des Ausdrucks in ein visuell lesbaren JSON- oder YAML-Format.

```
{{ output | to_nice_json }}  
{{ output | to_nice_yaml }}
```

Die Filter `from_json` und `from_yaml` setzen voraus, dass Zeichenfolgen im JSON- oder YAML-Format angegeben werden, um sie analysieren zu können.

```
{{ output | from_json }}  
{{ output | from_yaml }}
```

## VariablenTests

Die in Ansible-Playbooks in Kombination mit when-Klauseln verwendeten Ausdrücke sind Jinja2-Ausdrücke. Folgende integrierte Ansible-Tests werden zum Überprüfen der zurückgegebenen Werte verwendet: failed, changed, succeeded und skipped. Anhand der folgenden Aufgabe wird veranschaulicht, wie Tests innerhalb von bedingten Ausdrücken verwendet werden können.

```
tasks:  
  ...output omitted...  
  - debug: msg="the execution was aborted"  
    when: returnvalue is failed
```



### Literaturhinweise

#### **template – Templates a file out to a remote server – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/modules/template\\_module.html](https://docs.ansible.com/ansible/2.9/modules/template_module.html)

#### **Variables – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_variables.html](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_variables.html)

#### **Filter – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_filters.html](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_filters.html)

## ► Angeleitete Übung

# Bereitstellen benutzerdefinierter Dateien mit Jinja2-Vorlagen

In dieser Übung erstellen Sie eine einfache Vorlagendatei, die von Ihrem Playbook verwendet wird, um die benutzerdefinierte Datei „Nachricht des Tages“ auf jedem verwalteten Host zu installieren.

## Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Vorlage-Datei erstellen
- Verwenden der Vorlagendatei in einem Playbook.

## Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab file-template start` aus. Durch dieses Skript wird sichergestellt, dass Ansible auf `workstation` installiert ist, das Verzeichnis `/home/student/file-template` erstellt und die Datei `ansible.cfg` in dieses Verzeichnis heruntergeladen wird.

```
[student@workstation ~]$ lab file-template start
```



### Anmerkung

Alle in dieser Übung verwendeten Dateien sind zu Referenzzwecken auf `workstation` im Verzeichnis `/home/student/file-template/files` verfügbar.

## Anweisungen

- 1. Navigieren Sie auf `workstation` zum Arbeitsverzeichnis `/home/student/file-template`. Überprüfen Sie die Datei `inventory` im aktuellen Arbeitsverzeichnis. Diese Datei konfiguriert zwei Gruppen: `webservers` und `workstations`. Das System `servera.lab.example.com` befindet sich in der Gruppe `webservers` und das System `workstation.lab.example.com` in der Gruppe `workstations`.
- 1.1. Wechseln Sie in das Arbeitsverzeichnis `/home/student/file-template`.

```
[student@workstation ~]$ cd ~/file-template  
[student@workstation file-template]$
```

- 1.2. Zeigen Sie den Inhalt der Datei `inventory` an.

```
[webservers]
servera.lab.example.com

[workstations]
workstation.lab.example.com
```

- 2. Erstellen Sie eine Vorlage für die Nachricht des Tages, und fügen Sie sie in der Datei `motd.j2` im aktuellen Arbeitsverzeichnis ein. Beziehen Sie die folgenden Variablen in die Vorlage ein:

- `ansible_facts['fqdn']`, um den vollqualifizierten Domänennamen des verwalteten Hosts einzufügen.
- `ansible_facts['distribution']` und `ansible_facts['distribution_version']`, um Informationen über die Verteilung bereitzustellen.
- `system_owner` für die E-Mail des Systembesitzers. Diese Variable muss mit einem geeigneten Wert im Abschnitt `vars` der Playbook-Vorlage definiert sein.

```
This is the system {{ ansible_facts['fqdn'] }}.
This is a {{ ansible_facts['distribution'] }} version
{{ ansible_facts['distribution_version'] }} system.
Only use this system with permission.
Please report issues to: {{ system_owner }}.
```

- 3. Erstellen Sie eine Playbook-Datei mit der Bezeichnung `motd.yml` im aktuellen Arbeitsverzeichnis. Definieren Sie die Variable `system_owner` im Abschnitt `vars`, und beziehen Sie eine Aufgabe für das Modul `template` mit ein, die die Jinja2-Vorlage `motd.j2` der Remote-Datei `/etc/motd` auf den verwalteten Hosts zuordnet. Legen Sie den Besitzer und die Gruppe auf `root` und den Modus auf `0644` fest.

```
---
- name: configure SOE
  hosts: all
  remote_user: devops
  become: true
  vars:
    - system_owner: clyde@example.com
  tasks:
    - name: configure /etc/motd
      template:
        src: motd.j2
        dest: /etc/motd
        owner: root
        group: root
        mode: 0644
```

- 4. Führen Sie vor dem Ausführen des Playbooks den Befehl `ansible-playbook --syntax-check` aus, um die Syntax zu verifizieren. Wenn Fehler gemeldet werden,

## Kapitel 5 | Bereitstellen von Dateien auf verwalteten Hosts

korrigieren Sie sie, bevor Sie mit dem nächsten Schritt fortfahren. Es sollte eine Ausgabe ähnlich der folgenden angezeigt werden:

```
[student@workstation file-template]$ ansible-playbook --syntax-check motd.yml  
playbook: motd.yml
```

- 5. Führen Sie das Playbook `motd.yml` aus.

```
[student@workstation file-template]$ ansible-playbook motd.yml  
PLAY [all] ****  
  
TASK [Gathering Facts] ****  
ok: [servera.lab.example.com]  
ok: [workstation.lab.example.com]  
  
TASK [template] ****  
changed: [servera.lab.example.com]  
changed: [workstation.lab.example.com]  
  
PLAY RECAP ****  
servera.lab.example.com : ok=2    changed=1    unreachable=0    failed=0  
workstation.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```

- 6. Melden Sie sich auf `servera.lab.example.com` als der Benutzer `devops` an, um zu verifizieren, dass MOTD bei der Anmeldung richtig angezeigt wird. Melden Sie sich ab, wenn Sie fertig sind.

```
[student@workstation file-template]$ ssh devops@servera.lab.example.com  
This is the system servera.lab.example.com.  
This is a RedHat version 8.4 system.  
Only use this system with permission.  
Please report issues to: clyde@example.com.  
...output omitted...  
[devops@servera ~]# exit  
Connection to servera.lab.example.com closed.
```

## Beenden

Führen Sie den Befehl `lab file-template finish` aus, um nach der Übung eine Bereinigung durchzuführen.

```
[student@workstation ~]$ lab file-template finish
```

Hiermit ist die angeleitete Übung beendet.

## ► Praktische Übung

# Bereitstellen von Dateien auf verwalteten Hosts

### Performance-Checkliste

In diesem Lab führen Sie ein Playbook aus, das mithilfe einer Jinja2-Vorlage eine benutzerdefinierte Datei auf Ihre verwalteten Hosts erstellt.

### Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Vorlage-Datei erstellen
- Verwenden der Vorlagendatei in einem Playbook.

### Bevor Sie Beginnen

Melden Sie sich als student mit dem Passwort student bei workstation an.

Führen Sie auf workstation den Befehl `lab file-review start` aus. Dadurch wird sichergestellt, dass Ansible auf workstation installiert ist, das Verzeichnis `/home/student/file-review` erstellt und die Datei `ansible.cfg` in dieses Verzeichnis heruntergeladen wird. Zudem werden die Dateien `motd.yml`, `motd.j2`, `issue` und `inventory` in das Verzeichnis `/home/student/file-review/files` heruntergeladen.

```
[student@workstation ~]$ lab file-review start
```



#### Anmerkung

Alle in dieser Übung verwendeten Dateien sind auf workstation im Verzeichnis `/home/student/file-review/files` verfügbar.

### Anweisungen

1. Überprüfen Sie die Datei `inventory` im Verzeichnis `/home/student/file-review`. Diese Inventardatei definiert die Gruppe `servers`, die mit dem verwalteten Host `serverb.lab.example.com` verbunden ist.
2. Identifizieren Sie die Fakten auf `serverb.lab.example.com`, die den gesamten Arbeitsspeicher des Systems und die Anzahl der Prozessoren anzeigen.
3. Erstellen Sie eine Vorlage mit dem Namen `motd.j2` im aktuellen Verzeichnis für die Nachricht des Tages. Wenn sich der devops-Benutzer bei `serverb.lab.example.com` anmeldet, sollte eine Meldung mit dem gesamten Arbeitsspeicher und der Anzahl der Prozessoren angezeigt werden. Verwenden Sie die Fakten `ansible_facts['memtotal_mb']` und `ansible_facts['processor_count']`, um die Systemressourceninformationen für die Nachricht bereitzustellen.

## Kapitel 5 | Bereitstellen von Dateien auf verwalteten Hosts

4. Erstellen Sie eine neue Playbook-Datei mit dem Namen `motd.yml` im aktuellen Verzeichnis. Konfigurieren Sie mithilfe des Moduls `template` die zuvor erstellte Jinja2-Vorlagendatei `motd.j2`, um sie der Datei `/etc/motd` auf den verwalteten Hosts zuzuordnen. Diese Datei hat den Benutzer `root` als Besitzer und Gruppe und ihre Berechtigungen sind auf 0644 festgelegt. Erstellen Sie mit den Modulen `stat` und `debug` Aufgaben, um zu verifizieren, dass `/etc/motd` auf den verwalteten Hosts vorhanden ist und die Dateiinformationen für `/etc/motd` angibt. Verwenden Sie das Modul `copy`, um `files/issue` im Verzeichnis `/etc/` auf dem verwalteten Host zu platzieren, und verwenden Sie dabei die gleiche Eigentümerschaft und Berechtigungen wie `/etc/motd`. Verwenden Sie das Modul `file`, um sicherzustellen, dass `/etc/issue.net` eine symbolische Verknüpfung zu `/etc/issue` auf dem verwalteten Host ist. Konfigurieren Sie das Playbook so, dass es den Benutzer `devops` verwendet und den Parameter `become` auf `true` festlegt.
5. Führen Sie das Playbook in der Datei `motd.yml` aus.
6. Prüfen Sie, ob das Playbook in der Datei `motd.yml` korrekt ausgeführt wurde.

## Bewertung

Führen Sie auf `workstation` das Skript `lab file-review grade` aus, um den Erfolg dieser Übung zu bestätigen.

```
[student@workstation ~]$ lab file-review grade
```

## Beenden

Führen Sie auf `workstation` das Skript `lab file-review finish` aus, um nach der praktischen Übung eine Bereinigung durchzuführen.

```
[student@workstation ~]$ lab file-review finish
```

Hiermit ist die angeleitete Übung beendet.

## ► Lösung

# Bereitstellen von Dateien auf verwalteten Hosts

### Performance-Checkliste

In diesem Lab führen Sie ein Playbook aus, das mithilfe einer Jinja2-Vorlage eine benutzerdefinierte Datei auf Ihre verwalteten Hosts erstellt.

### Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Vorlage-Datei erstellen
- Verwenden der Vorlagendatei in einem Playbook.

### Bevor Sie Beginnen

Melden Sie sich als student mit dem Passwort student bei workstation an.

Führen Sie auf workstation den Befehl `lab file-review start` aus. Dadurch wird sichergestellt, dass Ansible auf workstation installiert ist, das Verzeichnis /home/student/file-review erstellt und die Datei ansible.cfg in dieses Verzeichnis heruntergeladen wird. Zudem werden die Dateien motd.yml, motd.j2, issue und inventory in das Verzeichnis /home/student/file-review/files heruntergeladen.

```
[student@workstation ~]$ lab file-review start
```



#### Anmerkung

Alle in dieser Übung verwendeten Dateien sind auf workstation im Verzeichnis /home/student/file-review/files verfügbar.

### Anweisungen

1. Überprüfen Sie die Datei `inventory` im Verzeichnis /home/student/file-review. Diese Inventardatei definiert die Gruppe servers, die mit dem verwalteten Host serverb.lab.example.com verbunden ist.
  - 1.1. Wechseln Sie auf workstation zum Verzeichnis /home/student/file-review.

```
[student@workstation ~]$ cd ~/file-review/
```

- 1.2. Zeigen Sie den Inhalt der Datei `inventory` an.

```
[servers]
serverb.lab.example.com
```

**Kapitel 5 |** Bereitstellen von Dateien auf verwalteten Hosts

2. Identifizieren Sie die Fakten auf `serverb.lab.example.com`, die den gesamten Arbeitsspeicher des Systems und die Anzahl der Prozessoren anzeigen.

Verwenden Sie das Modul `setup`, um eine Liste aller Fakten zum verwalteten Host `serverb.lab.example.com` zu erhalten. Die Fakten `ansible_processor_count` und `ansible_memtotal_mb` geben Auskunft über die Ressourcengrenzwerte des verwalteten Hosts.

```
[student@workstation file-review]$ ansible serverb.lab.example.com -m setup
serverb.lab.example.com | SUCCESS => {
    "ansible_facts": {
        ...output omitted...
    },
    "ansible_processor_count": 1,
    "ansible_memtotal_mb": 821,
    "changed": false
}
```

3. Erstellen Sie eine Vorlage mit dem Namen `motd.j2` im aktuellen Verzeichnis für die Nachricht des Tages. Wenn sich der `devops`-Benutzer bei `serverb.lab.example.com` anmeldet, sollte eine Meldung mit dem gesamten Arbeitsspeicher und der Anzahl der Prozessoren angezeigt werden. Verwenden Sie die Fakten `ansible_facts['memtotal_mb']` und `ansible_facts['processor_count']`, um die Systemressourceninformationen für die Nachricht bereitzustellen.

```
System total memory: {{ ansible_facts['memtotal_mb'] }} MiB.
System processor count: {{ ansible_facts['processor_count'] }}
```

4. Erstellen Sie eine neue Playbook-Datei mit dem Namen `motd.yml` im aktuellen Verzeichnis. Konfigurieren Sie mithilfe des Moduls `template` die zuvor erstellte Jinja2-Vorlagendatei `motd.j2`, um sie der Datei `/etc/motd` auf den verwalteten Hosts zuzuordnen. Diese Datei hat den Benutzer `root` als Besitzer und Gruppe und ihre Berechtigungen sind auf 0644 festgelegt. Erstellen Sie mit den Modulen `stat` und `debug` Aufgaben, um zu verifizieren, dass `/etc/motd` auf den verwalteten Hosts vorhanden ist und die Dateiinformationen für `/etc/motd` angibt. Verwenden Sie das Modul `copy`, um `files/issue` im Verzeichnis `/etc/` auf dem verwalteten Host zu platzieren, und verwenden Sie dabei die gleiche Eigentümerschaft und Berechtigungen wie `/etc/motd`. Verwenden Sie das Modul `file`, um sicherzustellen, dass `/etc/issue.net` eine symbolische Verknüpfung zu `/etc/issue` auf dem verwalteten Host ist. Konfigurieren Sie das Playbook so, dass es den Benutzer `devops` verwendet und den Parameter `become` auf `true` festlegt.

```
---
- name: Configure system
  hosts: all
  remote_user: devops
  become: true
  tasks:
    - name: Configure a custom /etc/motd
      template:
        src: motd.j2
        dest: /etc/motd
        owner: root
```

```
group: root
mode: 0644

- name: Check file exists
  stat:
    path: /etc/motd
  register: motd

- name: Display stat results
  debug:
    var: motd

- name: Copy custom /etc/issue file
  copy:
    src: files/issue
    dest: /etc/issue
    owner: root
    group: root
    mode: 0644

- name: Ensure /etc/issue.net is a symlink to /etc/issue
  file:
    src: /etc/issue
    dest: /etc/issue.net
    state: link
    owner: root
    group: root
    force: yes
```

5. Führen Sie das Playbook in der Datei `motd.yml` aus.

- 5.1. Führen Sie vor dem Ausführen des Playbooks den Befehl `ansible-playbook --syntax-check` aus, um die zugehörige Syntax zu verifizieren. Wenn Fehler gemeldet werden, korrigieren Sie sie, bevor Sie mit dem nächsten Schritt fortfahren. Es sollte eine Ausgabe ähnlich der folgenden angezeigt werden:

```
[student@workstation file-review]$ ansible-playbook --syntax-check motd.yml

playbook: motd.yml
```

- 5.2. Führen Sie das Playbook in der Datei `motd.yml` aus.

```
[student@workstation file-review]$ ansible-playbook motd.yml

PLAY [Configure system] ****
TASK [Gathering Facts] ****
ok: [serverb.lab.example.com]

TASK [Configure a custom /etc/motd] ****
changed: [serverb.lab.example.com]

TASK [Check file exists] ****
ok: [serverb.lab.example.com]
```

```
TASK [Display stat results] ****
ok: [serverb.lab.example.com] => {
    "motd": {
        "changed": false,
        "failed": false,
    ...output omitted...

TASK [Copy custom /etc/issue file] ****
changed: [serverb.lab.example.com]

TASK [Ensure /etc/issue.net is a symlink to /etc/issue] ****
changed: [serverb.lab.example.com]

PLAY RECAP ****
serverb.lab.example.com      : ok=6      changed=3      unreachable=0      failed=0
```

6. Prüfen Sie, ob das Playbook in der Datei `motd.yml` korrekt ausgeführt wurde.

Melden Sie sich bei `serverb.lab.example.com` als devops-Benutzer an, und stellen Sie sicher, dass bei der Anmeldung die Inhalte `/etc/motd` und `/etc/issue` angezeigt werden. Melden Sie sich ab, wenn Sie fertig sind.

```
[student@workstation file-review]$ ssh devops@serverb.lab.example.com
*----- PRIVATE SYSTEM -----*
*   Access to this computer system is restricted to authorised users only. *
*
*       Customer information is confidential and must not be disclosed. *
*-----*
System total memory: 821 MiB.
System processor count: 1
Activate the web console with: systemctl enable --now cockpit.socket

This system is not registered to Red Hat Insights. See https://cloud.redhat.com/
To register this system, run: insights-client --register

Last login: Thu Apr 25 22:09:33 2019 from 172.25.250.9
[devops@serverb ~]$ logout
```

## Bewertung

Führen Sie auf `workstation` das Skript `lab file-review grade` aus, um den Erfolg dieser Übung zu bestätigen.

```
[student@workstation ~]$ lab file-review grade
```

## Beenden

Führen Sie auf `workstation` das Skript `lab file-review finish` aus, um nach der praktischen Übung eine Bereinigung durchzuführen.

```
[student@workstation ~]$ lab file-review finish
```

Hiermit ist die angeleitete Übung beendet.

# Zusammenfassung

---

In diesem Kapitel wurden die folgenden Themen behandelt:

- Die Modul-Library `Files` enthält Module, mit denen Sie die meisten Dateiverwaltungsaufgaben erledigen können. Dazu zählen beispielsweise das Erstellen, Kopieren, Bearbeiten und Ändern von Berechtigungen und anderer Attribute von Dateien.
- Sie können Jinja2-Vorlagen verwenden, um Dateien für die Entwicklung dynamisch zu erstellen.
- Eine Jinja2-Vorlage besteht gewöhnlich aus zwei Elementen: Variablen und Ausdrücken. Diese Variablen und Ausdrücke werden durch Werte ersetzt, wenn die Jinja2-Vorlage gerendert ist.
- Mit Jinja2-Filtern werden Vorlagenausdrücke von Datenarten/Datenformaten in andere umgewandelt.

## Kapitel 6

# Verwalten komplexer Plays und Playbooks

### Ziel

Schreiben von Playbooks für größere, komplexere Plays und Playbooks.

### Ziele

- Ausgefeilte Host-Pattern erstellen, um Hosts für einen Play- oder Ad-hoc-Befehl effizient auszuwählen.
- Verwalten großer Playbooks durch Importieren oder Einbinden anderer Playbooks oder Aufgaben aus externen Dateien, entweder ohne Vorbedingungen oder auf der Grundlage eines bedingten Tests.

### Abschnitte

- Auswählen von Hosts mit Host-Pattern (und angeleitete Übung)
- Einbeziehen und Importieren von Dateien (und angeleitete Übung)

### Praktische Übung

- Verwalten komplexer Plays und Playbooks

# Auswählen von Hosts mit Host-Pattern

## Ziele

Am Ende dieses Abschnitts sollten Sie zu Folgendem in der Lage sein: Ausgefeilte Host-Pattern erstellen, um Hosts für ein Play oder einen Ad-hoc-Befehl effizient auszuwählen.

## Referenzieren von Inventarhosts

*Host-Pattern* werden verwendet, um die Hosts anzugeben, die Ziel eines Plays oder Ad-hoc-Befehls sind. In seiner einfachsten Form ist der Name eines verwalteten Hosts oder einer Hostgruppe im Inventar ein Host-Pattern, das diesen Host oder diese Hostgruppe angibt.

In diesem Kurs haben Sie bereits Host-Pattern verwendet. In einem Play gibt die Anweisung `hosts` die Hosts an, auf denen das Play ausgeführt werden soll. Geben Sie bei einem Ad-hoc-Befehl das Host-Pattern als Befehlszeilenargument für den Befehl `ansible` an.

Es ist normalerweise einfacher, zu steuern, welche Hosts ein Play als Ziel verwendet, indem Sie vorsichtig Host-Pattern und geeignete Inventargruppen verwenden, statt komplexe Bedingungen für die Aufgaben des Plays festzulegen. Sie müssen daher wissen, was Host-Pattern sind.

Das folgende Beispielinventar wird im gesamten Abschnitt verwendet, um Host-Pattern zu veranschaulichen.

```
[student@controlnode ~]$ cat myinventory
web.example.com
data.example.com

[lab]
labhost1.example.com
labhost2.example.com

[test]
test1.example.com
test2.example.com

[datacenter1]
labhost1.example.com
test1.example.com

[datacenter2]
labhost2.example.com
test2.example.com

[datacenter:children]
datacenter1
datacenter2
```

```
[new]
192.168.2.1
192.168.2.2
```

Um zu zeigen, wie Host-Pattern aufgelöst werden, führen Sie das Ansible-Playbook `playbook.yml` mit verschiedenen Host-Pattern aus, um unterschiedliche Teilmengen verwalteter Hosts aus diesem Beispielinventar zu verwenden.

## Verwaltete Hosts

Das einfachste Host-Pattern ist der Name eines einzelnen verwalteten Hosts, der im Inventar aufgeführt ist. Dieses Host-Pattern gibt an, dass der Host der einzige Host im Inventar ist, auf den der Befehl `ansible` ausgeführt wird.

Bei Ausführung des Playbooks soll die erste Aufgabe `Gathering Facts` für alle verwalteten Hosts ausgeführt werden, die dem Host-Pattern entsprechen. Wenn während dieser Aufgabe ein Fehler auftritt, kann dies dazu führen, dass der verwaltete Host aus dem Play entfernt wird.

Wenn im Inventar statt eines Hostnamens explizit eine IP-Adresse aufgeführt ist, kann sie als Host-Pattern verwendet werden. Wenn die IP-Adresse nicht im Inventar aufgeführt ist, kann sie nicht verwendet werden, um den Host anzugeben, selbst wenn die IP-Adresse im DNS in diesen Hostnamen aufgelöst wird.

Das folgende Beispiel zeigt, wie mithilfe eines Host-Pattern auf eine IP-Adresse in einem Inventar verwiesen werden kann.

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: 192.168.2.1
...output omitted...

[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ****
TASK [Gathering Facts] ****
ok: [192.168.2.1]
...output omitted...
```



### Anmerkung

Ein Problem mit Verweisen auf verwaltete Hosts anhand der IP-Adresse im Inventar besteht darin, dass es schwierig sein kann, sich zu merken, welche IP-Adresse zu welchem Host für Ihre Plays und Ad-hoc-Befehle gehört. Möglicherweise müssen Sie den Host jedoch für Verbindungszecke nach IP-Adresse angeben, wenn der Host keinen auflösbaren Hostnamen hat.

Mithilfe der Hostvariable `ansible_host` ist es möglich, dass ein Alias auf eine bestimmte IP-Adresse in Ihrem Inventar zeigt. Sie könnten in Ihrem Inventar z. B. über einen Host mit dem Namen `dummy.example` verfügen und Verbindungen dann mithilfe dieses Namens an die IP-Adresse 192.168.2.1 leiten, indem Sie die Datei `host_vars/dummy.example` erstellen, die die folgende Hostvariable enthält:

```
ansible_host: 192.168.2.1
```

## Angeben von Hosts mithilfe einer Gruppe

Sie haben bereits Inventarhostgruppen als Host-Pattern verwendet. Wenn ein Gruppenname als Host-Pattern verwendet wird, gibt dieses an, dass Ansible auf die Hosts angewendet wird, die Teil der Gruppe sind.

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: lab
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ****

TASK [Gathering Facts] ****
ok: [labhost1.example.com]
ok: [labhost2.example.com]
...output omitted...
```

Denken Sie daran, dass es die spezielle Gruppe `all` gibt, die allen verwalteten Hosts im Inventar entspricht.

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: all
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ****

TASK [Gathering Facts] ****
ok: [labhost2.example.com]
ok: [test2.example.com]
ok: [web.example.com]
ok: [data.example.com]
```

```
ok: [labhost1.example.com]
ok: [192.168.2.1]
ok: [test1.example.com]
ok: [192.168.2.2]
```

Es gibt auch eine spezielle Gruppe mit dem Namen `ungrouped`, die allen verwalteten Hosts im Inventar entspricht, die nicht Mitglied einer anderen Gruppe sind:

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: ungrouped
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ****
TASK [Gathering Facts] ****
ok: [web.example.com]
ok: [data.example.com]
```

## Abgleichen von mehreren Hosts mit Platzhaltern

Mit dem Platzhalterzeichen „`*`“ (Sternchen), das eine beliebige Zeichenfolge ersetzt, können Sie die gleiche Aktion ausführen wie mit dem Host-Pattern `all`. Wenn das Host-Pattern nur ein Sternchen in Anführungszeichen ist, stimmen alle Hosts im Inventar damit überein.

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: '*'
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ****
TASK [Gathering Facts] ****
ok: [labhost2.example.com]
ok: [test2.example.com]
ok: [web.example.com]
ok: [data.example.com]
ok: [labhost1.example.com]
ok: [192.168.2.1]
ok: [test1.example.com]
ok: [192.168.2.2]
```

## Wichtig

Einige Zeichen, die für Host-Pattern zur Verfügung stehen, können auch für die Shell verwendet werden. Dies kann ein Problem darstellen, wenn Host-Pattern verwendet werden, um mit `ansible` Ad-hoc-Befehle in der Befehlszeile auszuführen. Es ist eine empfohlene Vorgehensweise, die in der Befehlszeile verwendeten Host-Pattern in Anführungszeichen zu setzen, um sie vor unerwünschter Shell-Erweiterung zu schützen.

Wenn Sie in einem Ansible-Playbook spezielle Platzhalter oder Listenzeichenfolgen verwenden, müssen Sie ebenfalls das Host-Pattern in einfache Anführungszeichen setzen, um sicherzustellen, dass es korrekt analysiert wird.

```
---  
hosts: '!test1.example.com,development'
```

Das Sternchen kann außerdem verwendet werden, um verwaltete Hosts oder Gruppen abzugleichen, die eine bestimmte Teilzeichenfolge enthalten.

Das folgende Platzhalter-Host-Pattern entspricht z. B. allen Inventarnamen, die auf `.example.com` enden:

```
[student@controlnode ~]$ cat playbook.yml  
---  
- hosts: '*.example.com'  
...output omitted...  
[student@controlnode ~]$ ansible-playbook playbook.yml  
  
PLAY [Test Host Patterns] *****  
  
TASK [Gathering Facts] *****  
ok: [labhost1.example.com]  
ok: [test1.example.com]  
ok: [labhost2.example.com]  
ok: [test2.example.com]  
ok: [web.example.com]  
ok: [data.example.com]
```

Im folgenden Beispiel wird ein Platzhalter-Host-Pattern verwendet, das den Namen von Hosts oder Hostgruppen entspricht, die mit `192.168.2.` beginnen:

```
[student@controlnode ~]$ cat playbook.yml  
---  
- hosts: '192.168.2.*'  
...output omitted...  
[student@controlnode ~]$ ansible-playbook playbook.yml  
  
PLAY [Test Host Patterns] *****  
  
TASK [Gathering Facts] *****  
ok: [192.168.2.1]  
ok: [192.168.2.2]
```

Im nächsten Beispiel wird ein Platzhalter-Host-Pattern verwendet, das den Namen von Hosts oder Hostgruppen entspricht, die mit `datacenter` beginnen:

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: 'datacenter*'
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ****

TASK [Gathering Facts] ****
ok: [labhost1.example.com]
ok: [test1.example.com]
ok: [labhost2.example.com]
ok: [test2.example.com]
```



### Wichtig

Die Platzhalter-Host-Pattern ersetzen alle Inventarnamen, Hosts und Hostgruppen. Sie unterscheiden nicht zwischen Namen, die DNS-Namen, IP-Adressen oder Gruppen entsprechen, was zu unerwarteten Übereinstimmungen führen kann.

Vergleichen Sie basierend auf dem Beispielinventar z. B. die Ergebnisse bei Angabe des Host-Pattern `datacenter*` aus dem vorherigen Beispiel mit den Ergebnissen des Host-Pattern `data*`:

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: 'data*'
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ****

TASK [Gathering Facts] ****
ok: [labhost1.example.com]
ok: [test1.example.com]
ok: [labhost2.example.com]
ok: [test2.example.com]
ok: [data.example.com]
```

## Listen

Mithilfe logischer Listen lassen sich in einem Inventar mehrere Einträge referenzieren. Eine kommagetrennte Liste von Host-Pattern ersetzt alle Hosts, die einem dieser Host-Pattern entsprechen.

Wenn Sie eine kommagetrennte Liste verwalteter Hosts bereitstellen, werden alle diese verwalteten Hosts als Ziel verwendet:

## Kapitel 6 | Verwalten komplexer Plays und Playbooks

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: labhost1.example.com,test2.example.com,192.168.2.2
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ****

TASK [Gathering Facts] ****
ok: [labhost1.example.com]
ok: [test2.example.com]
ok: [192.168.2.2]
```

Wenn Sie eine kommagetrennte Liste von Gruppen bereitstellen, werden alle Hosts in diesen Gruppen als Ziel verwendet:

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: lab,datacenter1
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ****

TASK [Gathering Facts] ****
ok: [labhost1.example.com]
ok: [labhost2.example.com]
ok: [test1.example.com]
```

Sie können verwaltete Hosts, Hostgruppen und Platzhalter auch kombinieren, wie im Folgenden gezeigt:

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: lab,data*,192.168.2.2
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ****

TASK [Gathering Facts] ****
ok: [labhost1.example.com]
ok: [labhost2.example.com]
ok: [test1.example.com]
ok: [test2.example.com]
ok: [data.example.com]
ok: [192.168.2.2]
```



### Anmerkung

An Stelle eines Kommas kann der Doppelpunkt (:) verwendet werden. Das Komma ist jedoch das bevorzugte Trennzeichen, insbesondere wenn mit IPv6-Adressen als Namen verwalteter Hosts gearbeitet wird. Sie können die Doppelpunkt-Syntax in älteren Beispielen sehen.

Wenn ein Element in einer Liste mit einem kaufmännischen Und-Zeichen (&) beginnt, müssen Hosts mit diesem Element übereinstimmen, damit sie dem Host-Pattern entsprechen. Das Zeichen funktioniert ähnlich wie ein logisches AND.

Basierend auf unserem Beispielinventar entspricht das folgende Host-Pattern z. B. nur dann Rechner in der Gruppe `lab`, wenn sie auch in der Gruppe `datacenter1` enthalten sind:

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: lab,&datacenter1
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ****
TASK [Gathering Facts] ****
ok: [labhost1.example.com]
```

Mit den Host-Pattern `&lab`, `datacenter1` oder `datacenter1,&lab` könnten Sie auch angeben, dass Rechner in der Gruppe `datacenter1` nur dann dem jeweiligen Muster entsprechen, wenn sie in der Gruppe `lab` enthalten sind.

Sie können Hosts, die einem Pattern entsprechen, aus der Liste ausschließen, indem Sie das Ausrufungs- oder „Bang“-Zeichen (!) vor das Host-Pattern setzen. Dieses Zeichen funktioniert wie ein logisches NOT.

In diesem Beispiel werden alle Hosts abgeglichen, die in der Gruppe `datacenter` definiert sind, mit Ausnahme von `test2.example.com`, der auf dem Beispielinventar basiert:

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: datacenter,!test2.example.com
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ****
TASK [Gathering Facts] ****
ok: [labhost1.example.com]
ok: [test1.example.com]
ok: [labhost2.example.com]
```

Im vorherigen Beispiel könnte das Pattern '`!test2.example.com,datacenter`' verwendet werden, um denselben Effekt zu erzielen.

## Kapitel 6 | Verwalten komplexer Plays und Playbooks

Das letzte Beispiel veranschaulicht die Verwendung eines Host-Pattern, das allen Hosts im Testinventar entspricht, mit Ausnahme der verwalteten Hosts in der Gruppe `datacenter1`.

```
[student@controlnode ~]$ cat playbook.yml
---
- hosts: all,!datacenter1
...output omitted...
[student@controlnode ~]$ ansible-playbook playbook.yml

PLAY [Test Host Patterns] ****
TASK [Gathering Facts] ****
ok: [web.example.com]
ok: [data.example.com]
ok: [labhost2.example.com]
ok: [test2.example.com]
ok: [192.168.2.1]
ok: [192.168.2.2]
```



### Literaturhinweise

#### **Working with Patterns – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/user\\_guide/intro\\_patterns.html](https://docs.ansible.com/ansible/2.9/user_guide/intro_patterns.html)

#### **Working with Inventory – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/user\\_guide/intro\\_inventory.html](https://docs.ansible.com/ansible/2.9/user_guide/intro_inventory.html)

## ► Angeleitete Übung

# Auswählen von Hosts mit Host-Pattern

In dieser Übung untersuchen Sie, wie Host-Pattern verwendet werden, um Hosts aus dem Inventar für Plays oder Ad-hoc-Befehle anzugeben. Es werden mehrere Beispielinventare für die Untersuchung der Host-Pattern bereitgestellt.

## Ergebnisse

Sie sollten in der Lage sein, unterschiedliche Host-Pattern zum Zugriff auf verschiedene Hosts in einem Inventar zu verwenden.

## Bevor Sie Beginnen

Melden Sie sich als student mit dem Passwort student bei workstation an.

Führen Sie auf workstation den Befehl `lab projects-host start` aus. Das Skript erstellt das Projektverzeichnis `projects-host` und lädt die Ansible-Konfigurationsdatei und die Hostinventardatei herunter, die für die Übung erforderlich sind.

```
[student@workstation ~]$ lab projects-host start
```

## Anweisungen

- 1. Wechseln Sie auf `workstation` in das Arbeitsverzeichnis für die Übung `/home/student/projects-host`, und prüfen Sie den Inhalt des Verzeichnisses.

```
[student@workstation ~]$ cd ~/projects-host  
[student@workstation projects-host]$
```

- 1.1. Listen Sie den Inhalt dieses Verzeichnisses auf.

```
[student@workstation projects-host]$ ls  
ansible.cfg inventory1 inventory2 playbook.yml
```

- 1.2. Untersuchen Sie die Beispielinventardatei `inventory1`. Achten Sie darauf, wie das Inventar organisiert ist. Untersuchen Sie, welche Hosts und Gruppen sich im Inventar befinden und welche Domains verwendet werden.

```
srv1.example.com  
srv2.example.com  
s1.lab.example.com  
s2.lab.example.com
```

```
[web]  
jupiter.lab.example.com  
saturn.example.com
```

```
[db]
db1.example.com
db2.example.com
db3.example.com

[lb]
lb1.lab.example.com
lb2.lab.example.com

[boston]
db1.example.com
jupiter.lab.example.com
lb2.lab.example.com

[London]
db2.example.com
db3.example.com
file1.lab.example.com
lb1.lab.example.com

[dev]
web1.lab.example.com
db3.example.com

[stage]
file2.example.com
db2.example.com

[prod]
lb2.lab.example.com
db1.example.com
jupiter.lab.example.com

[function:children]
web
db
lb
city

[city:children]
boston
london
environments

[environments:children]
dev
stage
prod
new

[new]
172.25.252.23
172.25.252.44
172.25.252.32
```

- 1.3. Untersuchen Sie die Beispielinventardatei `inventory2`. Achten Sie darauf, wie das Inventar organisiert ist. Untersuchen Sie, welche Hosts und Gruppen sich im Inventar befinden und welche Domains verwendet werden.

```
workstation.lab.example.com

[london]
servera.lab.example.com

[berlin]
serverb.lab.example.com

[tokyo]
serverc.lab.example.com

[atlanta]
serverd.lab.example.com

[europe:children]
london
berlin
```

- 1.4. Überprüfen Sie zum Schluss den Inhalt des Playbooks `playbook.yml`. Beachten Sie, wie im Playbook das Modul `debug` verwendet wird, um die Namen aller verwalteten Hosts anzuzeigen.

```
---
- name: Resolve host patterns
  hosts:
    tasks:
      - name: Display managed host name
        debug:
          msg: "{{ inventory_hostname }}"
```

- 2. Ermitteln Sie mit einem Ad-hoc-Befehl, ob der Server `db1.example.com` in der Inventardatei `inventory1` enthalten ist.

```
[student@workstation projects-host]$ ansible db1.example.com -i inventory1 \
> --list-hosts
hosts (1):
  db1.example.com
```

- 3. Verweisen Sie mit einem Ad-hoc-Befehl auf eine IP-Adresse mit einem Host-Pattern, die im Inventar `inventory1` enthalten ist.

```
[student@workstation projects-host]$ ansible 172.25.252.44 -i inventory1 \
> --list-hosts
hosts (1):
  172.25.252.44
```

- 4. Verwenden Sie mit einem Ad-hoc-Befehl die Gruppe `all`, um alle verwalteten Hosts in der Inventardatei `inventory1` aufzulisten.

```
[student@workstation projects-host]$ ansible all -i inventory1 --list-hosts
hosts (17):
srv1.example.com
srv2.example.com
s1.lab.example.com
s2.lab.example.com
jupiter.lab.example.com
saturn.example.com
db1.example.com
db2.example.com
db3.example.com
lb1.lab.example.com
lb2.lab.example.com
file1.lab.example.com
web1.lab.example.com
file2.example.com
172.25.252.23
172.25.252.44
172.25.252.32
```

- 5. Verwenden Sie mit einem Ad-hoc-Befehl das Sternchen (\*), um alle Hosts aufzulisten, die in der Inventardatei `inventory1` auf `.example.com` enden.

```
[student@workstation projects-host]$ ansible '*.*.example.com' -i inventory1 \
> --list-hosts
hosts (14):
jupiter.lab.example.com
saturn.example.com
db1.example.com
db2.example.com
db3.example.com
lb1.lab.example.com
lb2.lab.example.com
file1.lab.example.com
web1.lab.example.com
file2.example.com
srv1.example.com
srv2.example.com
s1.lab.example.com
s2.lab.example.com
```

- 6. Wie Sie in der Ausgabe des vorherigen Befehls feststellen können, sind 14 Hosts in der Domain `*.example.com` enthalten. Ändern Sie das Host-Pattern im vorherigen Ad-hoc-Befehl so, dass Hosts in der Domain `*.lab.example.com` ignoriert werden.

```
[student@workstation projects-host]$ ansible '*.*.example.com, !*.lab.example.com' \
> -i inventory1 --list-hosts
hosts (7):
saturn.example.com
db1.example.com
db2.example.com
```

```
db3.example.com  
file2.example.com  
srv1.example.com  
srv2.example.com
```

- 7. Verwenden Sie ohne Zugreifen auf die Gruppen in der Inventardatei `inventory1` einen Ad-hoc-Befehl, um diese drei Hosts aufzulisten: `lb1.lab.example.com`, `s1.lab.example.com` und `db1.example.com`.

```
[student@workstation projects-host]$ ansible \  
> lb1.lab.example.com,s1.lab.example.com,db1.example.com -i inventory1 \  
> --list-hosts  
hosts (3):  
    lb1.lab.example.com  
    s1.lab.example.com  
    db1.example.com
```

- 8. Verwenden Sie ein Platzhalter-Host-Pattern in einem Ad-hoc-Befehl, um Hosts aufzulisten, deren IP-Adresse mit `172.25.` beginnt. IP-Adresse in der Inventardatei `inventory1`.

```
[student@workstation projects-host]$ ansible '172.25.*' -i inventory1 --list-hosts  
hosts (3):  
    172.25.252.23  
    172.25.252.44  
    172.25.252.32
```

- 9. Verwenden Sie ein Host-Pattern in einem Ad-hoc-Befehl, um alle Hosts aufzulisten, die in der Inventardatei `inventory1` mit dem Buchstaben „`s`“ beginnen.

```
[student@workstation projects-host]$ ansible 's*' -i inventory1 --list-hosts  
hosts (7):  
    saturn.example.com  
    srv1.example.com  
    srv2.example.com  
    s1.lab.example.com  
    s2.lab.example.com  
    file2.example.com  
    db2.example.com
```

Beachten Sie, dass die Hosts `file2.example.com` und `db2.example.com` in der Ausgabe des vorherigen Befehls enthalten sind. Beide Hosts werden in der Liste angezeigt, weil sie Mitglieder der Gruppe mit dem Namen `stage` sind, die ebenfalls mit dem Buchstaben „`s`“ beginnt.

- 10. Listen Sie mit einer Liste und Platzhalter-Host-Pattern in einem Ad-hoc-Befehl alle Hosts im Inventar `inventory1` in der Gruppe `prod` auf oder alle Hosts, deren IP-Adresse mit `172` beginnt oder deren Name die Zeichenfolge `lab` enthält.

```
[student@workstation projects-host]$ ansible 'prod,172*,*lab*' -i inventory1 \  
> --list-hosts  
hosts (11):  
    lb2.lab.example.com
```

```
db1.example.com
jupiter.lab.example.com
172.25.252.23
172.25.252.44
172.25.252.32
lb1.lab.example.com
file1.lab.example.com
web1.lab.example.com
s1.lab.example.com
s2.lab.example.com
```

- 11. Listen Sie mit einem Ad-hoc-Befehl alle Hosts auf, die sowohl zur Gruppe db als auch zur Gruppe london gehören.

```
[student@workstation projects-host]$ ansible 'db,&london' -i inventory1 \
> --list-hosts
hosts (2):
  db2.example.com
  db3.example.com
```

- 12. Ändern Sie den Wert hosts in der Datei `playbook.yml`, sodass alle Server in der Gruppe london angesprochen werden. Führen Sie das Playbook mit der Inventardatei `inventory2` aus.

```
...output omitted...
hosts: london
...output omitted...
```

```
[student@workstation projects-host]$ ansible-playbook -i inventory2 playbook.yml
...output omitted...
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]
...output omitted...
```

- 13. Ändern Sie den Wert hosts in der Datei `playbook.yml`, sodass alle Server in der verschachtelten Gruppe europe angesprochen werden. Führen Sie das Playbook mit der Inventardatei `inventory2` aus.

```
...output omitted...
hosts: europe
...output omitted...
```

```
[student@workstation projects-host]$ ansible-playbook -i inventory2 playbook.yml
...output omitted...
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]
ok: [serverb.lab.example.com]
...output omitted...
```

- 14. Ändern Sie den Wert hosts in der Datei `playbook.yml`, sodass alle Server, die nicht zu einer Gruppe gehören, angesprochen werden. Führen Sie das Playbook mit der Inventardatei `inventory2` aus.

```
...output omitted...
hosts: ungrouped
...output omitted...
```

```
[student@workstation projects-hosts]$ ansible-playbook -i inventory2 playbook.yml
...output omitted...
TASK [Gathering Facts] ****
ok: [workstation.lab.example.com]
...output omitted...
```

## Beenden

Führen Sie auf `workstation` das Skript `lab projects-host finish` aus, um diese Übung zu bereinigen.

```
[student@workstation ~]$ lab projects-host finish
```

Hiermit ist die angeleitete Übung beendet.

# Einbeziehen und Importieren von Dateien

## Ziele

Am Ende dieses Abschnitts sollten Sie zu Folgendem in der Lage sein: Große Playbooks durch Importieren oder Einbeziehen anderer Playbooks oder Aufgaben aus externen Dateien, entweder ohne Bedingungen oder auf Basis eines Bedingungstests, verwalten.

## Verwalten großer Playbooks

Wenn ein Playbook lang oder komplex wird, können Sie es in kleinere Dateien aufteilen, damit es einfacher verwaltet werden kann. Sie können mehrere Playbooks modular in einem Haupt-Playbook kombinieren oder Aufgabenlisten aus einer Datei in ein Play einfügen. Dies kann die Wiederverwendung von Plays oder Aufgabensequenzen in verschiedenen Projekten erleichtern.

## Einbeziehen oder Importieren von Dateien

Ansible kann Inhalt in ein Playbook mithilfe von zwei Vorgängen aufnehmen. Sie können Inhalt *einbeziehen* oder *importieren*.

Das Einbeziehen von Inhalt ist ein *dynamischer* Vorgang. Ansible verarbeitet einbezogenen Inhalt bei der Ausführung von Playbooks, sobald der Inhalt erreicht wird.

Das Importieren von Inhalt ist ein *statischer* Vorgang. Ansible verarbeitet den importierten Inhalt vor, wenn das Playbook vor der Ausführung analysiert wird.

## Importieren von Playbooks

Mit der Anweisung `import_playbook` können Sie externe Dateien mit Play-Listen in ein Playbook importieren. Mit anderen Worten: Sie können über ein Master-Playbook verfügen, das eines oder mehrere zusätzliche Playbooks importiert.

Da der zu importierende Inhalt ein vollständiges Playbook ist, kann `import_playbook` nur auf der obersten Ebene eines Playbooks und nicht innerhalb eines Plays verwendet werden. Wenn Sie mehrere Playbooks importieren, werden diese importiert und der Reihe nach ausgeführt.

Ein einfaches Beispiel für ein Master-Playbook, das zwei zusätzliche Playbooks importiert, wird unten gezeigt:

```
- name: Prepare the web server
  import_playbook: web.yml

- name: Prepare the database server
  import_playbook: db.yml
```

Sie können mit importierten Playbooks auch Plays in Ihrem Master-Playbook verschachteln.

```
- name: Play 1
  hosts: localhost
  tasks:
    - debug:
        msg: Play 1

  - name: Import Playbook
    import_playbook: play2.yml
```

Im vorhergehenden Beispiel wird Play 1 zuerst ausgeführt. Anschließend werden die Plays aus dem importierten Playbook play2.yml ausgeführt.

## Importieren und Einbeziehen von Aufgaben

Sie können eine Liste mit Aufgaben aus einer Aufgabendatei in ein Play importieren oder einbeziehen. Eine Aufgabendatei ist eine Datei, die eine Liste mit Aufgaben enthält:

```
[admin@node ~]$ cat webserver_tasks.yml
- name: Installs the httpd package
  yum:
    name: httpd
    state: latest

- name: Starts the httpd service
  service:
    name: httpd
    state: started
```

## Importieren von Aufgabendateien

Sie können eine Aufgabendatei mit `import_tasks` statisch in ein Play in einem Playbook importieren. Wenn Sie eine Aufgabendatei importieren, werden die Aufgaben in dieser Datei direkt eingefügt, wenn das Playbook analysiert wird. Die Position von `import_tasks` im Playbook steuert, wo die Aufgaben eingefügt und in welcher Reihenfolge mehrere Importe ausgeführt werden.

```
---
- name: Install web server
  hosts: webservers
  tasks:
    - import_tasks: webserver_tasks.yml
```

Wenn Sie eine Aufgabendatei importieren, werden die Aufgaben in dieser Datei direkt eingefügt, wenn das Playbook analysiert wird. Weil `import_tasks` die Aufgaben bei der Analyse des Playbooks statisch importiert, hat das Auswirkungen auf die Funktionsweise.

- Bei Verwendung von `import_tasks` werden bedingte Anweisungen wie `when`, die für den Import festgelegt sind, auf jede der importierten Aufgaben angewendet.
- Sie können mit `import_tasks` keine Schleifen verwenden.
- Wenn Sie den Namen der zu importierenden Datei mit einer Variablen angeben, können Sie keine Host- oder Gruppeninventarvariable verwenden.

## Einbeziehen von Aufgabendateien

Sie können eine Aufgabendatei mit `include_tasks` auch dynamisch in ein Play in einem Playbook einbeziehen.

```
---
```

```
- name: Install web server
  hosts: webservers
  tasks:
    - include_tasks: webserver_tasks.yml
```

`include_tasks` verarbeitet den Inhalt im Playbook erst, wenn das Play ausgeführt und dieser Teil des Plays erreicht wird. Die Reihenfolge, in der Playbook-Inhalte verarbeitet werden, wirkt sich auf die Funktionsweise der Funktion „Aufgaben einschließen“ aus.

- Bei Verwendung von `include_tasks` bestimmen bedingte Anweisungen wie `when`, die für die einzubeziehenden Aufgaben festgelegt sind, ob die Aufgaben überhaupt in das Play einbezogen werden oder nicht.
- Wenn Sie `ansible-playbook --list-tasks` ausführen, um die Aufgaben im Playbook aufzulisten, werden Aufgaben in den einbezogenen Aufgabendateien nicht angezeigt. Die Aufgaben, die die Aufgabendateien einbeziehen, werden angezeigt. Im Vergleich dazu würde `import_tasks` keine Aufgaben auflisten, die Aufgabendateien importieren, sondern die einzelnen Aufgaben aus den importierten Aufgabendateien.
- Sie können `ansible-playbook --start-at-task` nicht verwenden, um die Playbook-Ausführung von einer Aufgabe aus zu starten, die sich in einer einbezogenen Aufgabendatei befindet.
- Sie können mit der Anweisung `notify` keinen Handler-Namen auslösen, der sich in einer einbezogenen Aufgabendatei befindet. Im Haupt-Playbook können Sie einen Handler auslösen, der eine gesamte Aufgabendatei enthält. In diesem Fall werden alle Aufgaben in der einbezogenen Datei ausgeführt.



### Anmerkung

Eine ausführlichere Erläuterung der Verhaltensunterschiede von `import_tasks` und `include_tasks`, wenn Bedingungen verwendet werden, finden Sie unter „Bedingungen“ [[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_conditionals.html#applying-when-to-roles-imports-and-includes](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_conditionals.html#applying-when-to-roles-imports-and-includes)] im *Ansible-Benutzerhandbuch*.

## Anwendungsfälle für Aufgabendateien

In den folgenden Situationen kann es sinnvoll sein, Gruppen von Aufgaben als vom Playbook getrennte externe Dateien zu verwalten:

- Wenn für neue Server eine vollständige Konfiguration erforderlich ist, können Administratoren verschiedene Gruppen von Aufgaben zum Anlegen neuer Benutzer, Installieren von Paketen, Konfigurieren von Services, Konfigurieren von Berechtigungen, Einrichten des Zugangs zum gemeinsam genutzten Dateisystem, Verstärken von Servern, Installieren von Sicherheitsupdates und Installieren eines Überwachungs-Agenten erstellen. Jede dieser Gruppen von Aufgaben könnte über eine separate, eigenständige Aufgabendatei verwaltet werden.
- Wenn Server von Entwicklern, Systemadministratoren und Datenbankadministratoren gemeinsam verwaltet werden, dann können von jedem Fachgebiet eigene Aufgabendateien

geschrieben werden, die anschließend vom Systemmanager überprüft und integriert werden können.

- Wenn für einen Server eine besondere Konfiguration erforderlich ist, kann diese als Gruppe von Aufgaben integriert werden, die auf Basis einer Bedingung ausgeführt wird. Anders gesagt, werden die Aufgaben nur dann eingeschlossen, wenn bestimmte Kriterien erfüllt sind.
- Wenn eine Gruppe von Servern eine bestimmte Aufgabe oder Aufgabengruppe ausführen muss, können die Aufgaben vielleicht nur auf einem Server ausgeführt werden, der Teil einer bestimmten Hostgruppe ist.

## Verwalten von Aufgabendateien

Sie können ein spezielles Verzeichnis für Aufgabendateien anlegen und alle Aufgabendateien in diesem Verzeichnis speichern. Ihr Playbook kann dann einfach Aufgabendateien aus diesem Verzeichnis einbeziehen oder importieren. Dadurch kann ein komplexes Playbook erstellt und die Verwaltung seiner Struktur und Komponenten vereinfacht werden.

## Definieren von Variablen für externe Plays und Aufgaben

Die Einbeziehung von Plays oder Aufgaben aus externen Dateien in Playbooks mit den Import- und Include-Funktionen von Ansible verbessert die Möglichkeit, Aufgaben und Playbooks in einer Ansible-Umgebung wiederzuverwenden. Um die potenzielle Wiederverwendung zu erhöhen, sollten diese Aufgaben- und Play-Dateien so allgemein wie möglich gehalten werden. Mit Variablen können Play- und Aufgabenelemente parametrisiert werden, um den Anwendungsbereich von Aufgaben und Plays zu erweitern.

Die folgende Aufgabendatei installiert beispielsweise das für einen Webservice erforderliche Paket und aktiviert und startet dann den erforderlichen Service.

```
---  
- name: Install the httpd package  
  yum:  
    name: httpd  
    state: latest  
- name: Start the httpd service  
  service:  
    name: httpd  
    enabled: true  
    state: started
```

Wenn Sie die Paket- und Serviceelemente wie im folgenden Beispiel parametrisieren, kann die Aufgabendatei auch für die Installation und Verwaltung anderer Software und ihrer Services verwendet werden, anstatt nur für Webservices.

```
---  
- name: Install the {{ package }} package  
  yum:  
    name: "{{ package }}"  
    state: latest  
- name: Start the {{ service }} service  
  service:
```

```
name: "{{ service }}"
enabled: true
state: started
```

Wenn Sie die Aufgabendatei in ein Playbook integrieren, definieren Sie die für die Aufgabenausführung zu verwendenden Variablen wie folgt:

```
...output omitted...
tasks:
- name: Import task file and set variables
  import_tasks: task.yml
vars:
  package: httpd
  service: httpd
```

Ansible sorgt dafür, dass die übergebenen Variablen für die aus der externen Datei importierten Aufgaben verfügbar sind.

Sie können dieselbe Technik verwenden, um die Wiederverwendbarkeit von Play-Dateien zu erhöhen. Bei der Integration einer Play-Datei übergeben Sie die Variablen, die für die Ausführung des Plays verwendet werden sollen, wie folgt:

```
...output omitted...
- name: Import play file and set the variable
  import_playbook: play.yml
vars:
  package: mariadb
```



### Wichtig

In früheren Versionen von Ansible wurde eine `include`-Funktion verwendet, um je nach Kontext sowohl Playbooks als auch Aufgabendateien einzubeziehen. Diese Funktion ist aus verschiedenen Gründen veraltet.

Vor Ansible 2.0 funktionierte `include` wie ein statischer Import. In Ansible 2.0 wurde dies in eine dynamische Vorgehensweise geändert, was jedoch zu Einschränkungen führte. In Ansible 2.1 wurde ermöglicht, dass `include` abhängig von den Aufgabeneinstellungen dynamisch oder statisch sein konnte, was jedoch verwirrend und fehleranfällig war. Außerdem gab es Probleme sicherzustellen, dass `include` in allen Kontexten korrekt funktioniert.

Deswegen wurde `include` in Ansible 2.4 durch neue Anweisungen wie `include_tasks`, `import_tasks` und `import_playbook` ersetzt. Sie finden möglicherweise Beispiele für `include` in älteren Playbooks, aber Sie sollten diese nicht in neuen Playbooks verwenden.



### Literaturhinweise

#### **Including and Importing – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_reuse\\_includes.html](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_reuse_includes.html)

#### **Creating Reusable Playbooks – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_reuse.html](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_reuse.html)

#### **Conditionals – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_conditionals.html](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_conditionals.html)

## ► Angeleitete Übung

# Einbeziehen und Importieren von Dateien

In dieser Übung beziehen Sie Playbooks und Aufgaben in ein Ansible-Playbook der obersten Ebene ein bzw. importieren sie.

## Ergebnisse

Sie können Aufgaben- und Playbook-Dateien in Playbooks einbeziehen.

## Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab projects-file start` aus. Das Skript erstellt das Arbeitsverzeichnis `/home/student/projects-file` sowie zugehörige Projektdateien.

```
[student@workstation ~]$ lab projects-file start
```

## Anweisungen

- 1. Wechseln Sie auf `workstation` als Benutzer `student` in das Verzeichnis `/home/student/projects-file`.

```
[student@workstation ~]$ cd ~/projects-file  
[student@workstation projects-file]$
```

- 2. Prüfen Sie den Inhalt der drei Dateien im Unterverzeichnis `tasks`.

- 2.1. Prüfen Sie den Inhalt der Datei `tasks/environment.yml`. Die Datei enthält Aufgaben zur Paketinstallation und Serviceverwaltung.

```
---  
- name: Install the {{ package }} package  
  yum:  
    name: "{{ package }}"  
    state: latest  
- name: Start the {{ service }} service  
  service:  
    name: "{{ service }}"  
    enabled: true  
    state: started
```

- 2.2. Prüfen Sie den Inhalt der Datei `tasks/firewall.yml`. Die Datei enthält Aufgaben zur Installation, Verwaltung und Konfiguration von Firewall-Software.

```
---
```

- name: Install the firewall
  - yum:
    - name: "{{ firewall\_pkg }}"
    - state: latest
  - name: Start the firewall
    - service:
      - name: "{{ firewall\_svc }}"
      - enabled: true
      - state: started
  - name: Open the port for {{ rule }}
    - firewalld:
      - service: "{{ item }}"
      - immediate: true
      - permanent: true
      - state: enabled
    - loop: "{{ rule }}"

- 2.3. Prüfen Sie den Inhalt der Datei `tasks/placeholder.yml`. Die Datei enthält eine Aufgabe zum Füllen einer Platzhalter-Webinhaltsdatei.

```
---
```

- name: Create placeholder file
  - copy:
    - content: "{{ ansible\_facts['fqdn'] }} has been customized using Ansible.\n"
    - dest: "{{ file }}"

- 3. Prüfen Sie den Inhalt der Datei `test.yml` im Unterverzeichnis `plays`. Diese Datei enthält ein Play, das Verbindungen zu einem Webservice testet.

```
---
```

- name: Test web service
  - hosts: localhost
  - become: no
  - tasks:
    - name: connect to internet web server
      - uri:
        - url: "{{ url }}"
        - status\_code: 200

- 4. Erstellen Sie ein Playbook mit dem Namen `playbook.yml`. Legen Sie für das erste Play den Namen `Configure web server` fest. Das Play sollte für den verwalteten Host `servera.lab.example.com` ausgeführt werden, der in der Datei `inventory` definiert ist. Der Anfang der Datei sollte folgendermaßen aussehen:

```
---
```

- name: Configure web server
  - hosts: servera.lab.example.com

- 5. Definieren Sie im Playbook `playbook.yml` den Aufgabenabschnitt mit drei Aufgabengruppen. Beziehen Sie den ersten Satz von Aufgaben aus der Aufgabendatei `tasks/environment.yml` ein. Definieren Sie die erforderlichen Variablen für die Installation des `httpd`-Pakets und zum Aktivieren und Starten des `httpd`-Services. Importieren Sie den zweiten Satz von Aufgaben aus der Aufgabendatei `tasks/firewall.yml`. Definieren Sie die erforderlichen Variablen für die Installation des `firewalld`-Pakets zum Aktivieren und Starten des `firewall`-Services und zum Zulassen von `http`-Verbindungen. Importieren Sie die dritte Aufgabegruppe aus der Aufgabendatei `tasks/placeholder.yml`.
- 5.1. Erstellen Sie den Aufgabenabschnitt im ersten Play, indem Sie dem Playbook `playbook.yml` den folgenden Eintrag hinzufügen.

```
tasks:
```

- 5.2. Beziehen Sie den ersten Satz von Aufgaben aus `tasks/environment.yml` mit der Funktion `include_tasks` ein. Legen Sie die Variablen `package` und `service` auf `httpd` fest.

```
- name: Include the environment task file and set the variables
  include_tasks: tasks/environment.yml
  vars:
    package: httpd
    service: httpd
```

- 5.3. Importieren Sie mit `import_tasks` die zweite Aufgabengruppe aus `tasks/firewall.yml`. Setzen Sie die Variablen `firewall_pkg` und `firewall_svc` auf `firewalld`. Legen Sie die Variable `rule` auf `http` fest.

```
- name: Import the firewall task file and set the variables
  import_tasks: tasks/firewall.yml
  vars:
    firewall_pkg: firewalld
    firewall_svc: firewalld
    rule:
      - http
      - https
```

- 5.4. Importieren Sie mit `import_tasks` die letzte Aufgabengruppe aus `tasks/placeholder.yml`. Legen Sie die Variable `file` auf `/var/www/html/index.html` fest.

```
- name: Import the placeholder task file and set the variable
  import_tasks: tasks/placeholder.yml
  vars:
    file: /var/www/html/index.html
```

- 6. Fügen Sie dem Playbook `playbook.yml` ein zweites und letztes Play mit dem Inhalt des Playbooks `plays/test.yml` hinzu.
- 6.1. Fügen Sie dem Playbook `playbook.yml` ein zweites Play hinzu, um die Webserverinstallation zu überprüfen. Importieren Sie das Play aus `plays/test.yml`. Legen Sie die Variable `url` auf `http://servera.lab.example.com` fest.

```
- name: Import test play file and set the variable
  import_playbook: plays/test.yml
  vars:
    url: 'http://servera.lab.example.com'
```

6.2. Ihr Playbook sollte nach Abschluss der Änderungen wie folgt aussehen:

```
---
- name: Configure web server
  hosts: servera.lab.example.com

  tasks:
    - name: Include the environment task file and set the variables
      include_tasks: tasks/environment.yml
      vars:
        package: httpd
        service: httpd

    - name: Import the firewall task file and set the variables
      import_tasks: tasks/firewall.yml
      vars:
        firewall_pkg: firewalld
        firewall_svc: firewalld
        rule:
          - http
          - https

    - name: Import the placeholder task file and set the variable
      import_tasks: tasks/placeholder.yml
      vars:
        file: /var/www/html/index.html

- name: Import test play file and set the variable
  import_playbook: plays/test.yml
  vars:
    url: 'http://servera.lab.example.com'
```

6.3. Speichern Sie die Änderungen am Playbook `playbook.yml`.

- ▶ 7. Verifizieren Sie vor Ausführung des Playbooks, dass dessen Syntax korrekt ist, indem Sie `ansible-playbook --syntax-check` ausführen. Wenn Fehler gemeldet werden, korrigieren Sie diese, bevor Sie mit dem nächsten Schritt fortfahren.

```
[student@workstation projects-file]$ ansible-playbook playbook.yml --syntax-check
playbook: playbook.yml
```

- ▶ 8. Führen Sie das Playbook `playbook.yml` aus. Die Ausgabe des Playbooks zeigt den Import der Aufgaben- und Play-Dateien.

```
[student@workstation projects-file]$ ansible-playbook playbook.yml

PLAY [Configure web server] ****

TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Install the httpd package] ****
changed: [servera.lab.example.com]

TASK [Start the httpd service] ****
changed: [servera.lab.example.com]

TASK [Install the firewall] ****
ok: [servera.lab.example.com]

TASK [Start the firewall] ****
ok: [servera.lab.example.com]

TASK [Open the port for ['http', 'https']] ****
changed: [servera.lab.example.com] => (item=http)
changed: [servera.lab.example.com] => (item=https)

TASK [Create placeholder file] ****
changed: [servera.lab.example.com]

PLAY [Test web service] ****

TASK [Gathering Facts] ****
ok: [localhost]

TASK [connect to internet web server] ****
ok: [localhost]

PLAY RECAP ****
localhost                  : ok=2      changed=0      unreachable=0      failed=0
servera.lab.example.com   : ok=8      changed=4      unreachable=0      failed=0
```

## Beenden

Führen Sie auf workstation das Skript `lab projects-file finish` aus, um diese Übung zu bereinigen.

```
[student@workstation ~]$ lab projects-file finish
```

Hiermit ist die angeleitete Übung beendet.

## ► Praktische Übung

# Verwalten komplexer Plays und Playbooks

### Performance-Checkliste

In dieser Übung passen Sie ein komplexes Playbook so an, dass es mit Host-Pattern, Includes und Importen einfacher verwaltet werden kann.

### Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Vereinfachen von Hostreferenzen in einem Playbook durch Angabe von Hostmusters
- Restrukturieren eines Playbooks, sodass Aufgaben aus externen Aufgabendateien importiert werden

### Bevor Sie Beginnen

Melden Sie sich als student mit dem Passwort student bei workstation an.

Führen Sie auf workstation den Befehl `lab projects-review start` aus. Dieses Setup-Skript stellt sicher, dass die verwalteten Hosts im Netzwerk erreichbar sind. Dies gewährleistet auch die Installation der korrekten Ansible-Konfigurationsdatei, der Inventardatei und des Playbooks auf dem Kontrollknoten im Verzeichnis `/home/student/projects-review`.

```
[student@workstation ~]$ lab projects-review start
```

### Anweisungen

Sie haben ein Playbook vom vorherigen Administrator übernommen. Das Playbook wird verwendet, um den Webservice auf `servera.lab.example.com`, `serverb.lab.example.com`, `serverc.lab.example.com` und `serverd.lab.example.com` zu konfigurieren. Das Playbook konfiguriert auch die Firewall auf den vier verwalteten Hosts so, dass Webdatenverkehr zugelassen wird.

Nehmen Sie die folgenden Änderungen an der Playbook-Datei `playbook.yml` vor, damit sie einfacher zu verwalten ist.

1. Vereinfachen Sie die Liste der verwalteten Hosts im Playbook `/home/student/projects-review/playbook.yml` mithilfe eines Platzhalter-Host-Pattern.
2. Strukturieren Sie das Playbook so um, dass die ersten drei Aufgaben im Playbook in einer externen Aufgabendatei unter `tasks/web_tasks.yml` gespeichert werden. Verwenden Sie `import_tasks`, um diese Aufgabendatei in das Playbook zu integrieren.
3. Strukturieren Sie das Playbook so um, dass die vierte, fünfte und sechste Aufgabe im Playbook in einer externen Aufgabendatei unter `tasks/firewall_tasks.yml` gespeichert werden. Verwenden Sie `import_tasks`, um diese Aufgabendatei in das Playbook zu integrieren.

## Kapitel 6 | Verwalten komplexer Plays und Playbooks

4. Zwischen den Dateien `tasks/web_tasks.yml` und `tasks/firewall_tasks.yml` gibt es einige doppelte Aufgaben. Verschieben Sie die Aufgaben, mit denen Pakete installiert und Services aktiviert werden, in eine neue Datei mit dem Namen `tasks/install_and_enable.yml`, und aktualisieren Sie sie, um Variablen zu verwenden. Ersetzen Sie die ursprünglichen Aufgaben durch `import_tasks`-Anweisungen, die entsprechende Variablenwerte übergeben.
5. Verifizieren Sie, dass die Änderungen am Playbook `playbook.yml` korrekt vorgenommen wurden, und führen Sie dann das Playbook aus.

## Bewertung

Führen Sie auf `workstation` den Befehl `lab projects-review grade` aus, um den Erfolg dieser Übung zu bestätigen. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie das Skript so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab projects-review grade
```

## Beenden

Führen Sie auf `workstation` das Skript `lab projects-review finish` aus, um die in dieser praktischen Übung erstellten Ressourcen zu bereinigen.

```
[student@workstation ~]$ lab projects-review finish
```

Hiermit ist die praktische Übung abgeschlossen.

## ► Lösung

# Verwalten komplexer Plays und Playbooks

### Performance-Checkliste

In dieser Übung passen Sie ein komplexes Playbook so an, dass es mit Host-Pattern, Includes und Importen einfacher verwaltet werden kann.

### Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Vereinfachen von Hostreferenzen in einem Playbook durch Angabe von Hostmusters
- Restrukturieren eines Playbooks, sodass Aufgaben aus externen Aufgabendateien importiert werden

### Bevor Sie Beginnen

Melden Sie sich als student mit dem Passwort student bei workstation an.

Führen Sie auf workstation den Befehl `lab projects-review start` aus. Dieses Setup-Skript stellt sicher, dass die verwalteten Hosts im Netzwerk erreichbar sind. Dies gewährleistet auch die Installation der korrekten Ansible-Konfigurationsdatei, der Inventardatei und des Playbooks auf dem Kontrollknoten im Verzeichnis `/home/student/projects-review`.

```
[student@workstation ~]$ lab projects-review start
```

### Anweisungen

Sie haben ein Playbook vom vorherigen Administrator übernommen. Das Playbook wird verwendet, um den Webservice auf `servera.lab.example.com`, `serverb.lab.example.com`, `serverc.lab.example.com` und `serverd.lab.example.com` zu konfigurieren. Das Playbook konfiguriert auch die Firewall auf den vier verwalteten Hosts so, dass Webdatenverkehr zugelassen wird.

Nehmen Sie die folgenden Änderungen an der Playbook-Datei `playbook.yml` vor, damit sie einfacher zu verwalten ist.

1. Vereinfachen Sie die Liste der verwalteten Hosts im Playbook `/home/student/projects-review/playbook.yml` mithilfe eines Platzhalter-Host-Pattern.
  - 1.1. Wechseln Sie in das Arbeitsverzeichnis `/home/student/projects-review`. Überprüfen Sie den Parameter `hosts` in der Datei `playbook.yml`.

```
[student@workstation ~]$ cd ~/projects-review
[student@workstation projects-review]$ cat playbook.yml
---
- name: Install and configure web service
```

```
hosts:  
  - servera.lab.example.com  
  - serverb.lab.example.com  
  - serverc.lab.example.com  
  - serverd.lab.example.com  
...output omitted...
```

- 1.2. Verifizieren Sie, dass das Host-Pattern `server*.lab.example.com` die vier verwalteten Hosts, die Ziel des Playbooks `playbook.yml` sind, korrekt identifiziert.

```
[student@workstation projects-review]$ ansible server*.lab.example.com \  
> --list-hosts  
hosts (4):  
  servera.lab.example.com  
  serverb.lab.example.com  
  serverc.lab.example.com  
  serverd.lab.example.com
```

- 1.3. Ersetzen Sie die Hostliste im Playbook `playbook.yml` durch das Host-Pattern `server*.lab.example.com`.

```
---  
- name: Install and configure web service  
  hosts: server*.lab.example.com  
...output omitted...
```

2. Strukturieren Sie das Playbook so um, dass die ersten drei Aufgaben im Playbook in einer externen Aufgabendatei unter `tasks/web_tasks.yml` gespeichert werden. Verwenden Sie `import_tasks`, um diese Aufgabendatei in das Playbook zu integrieren.

- 2.1. Erstellen Sie das Unterverzeichnis `tasks`.

```
[student@workstation projects-review]$ mkdir tasks
```

- 2.2. Platzieren Sie den Inhalt der ersten drei Aufgaben im Playbook `playbook.yml` in der Datei `tasks/web_tasks.yml`. Die Aufgabendatei sollte den folgenden Inhalt haben:

```
---  
- name: Install httpd  
  yum:  
    name: httpd  
    state: latest  
  
- name: Enable and start httpd  
  service:  
    name: httpd  
    enabled: true  
    state: started  
  
- name: Tuning configuration installed  
  copy:  
    src: files/tune.conf
```

```
dest: /etc/httpd/conf.d/tune.conf
owner: root
group: root
mode: 0644
notify:
  - restart httpd
```

- 2.3. Entfernen Sie die ersten drei Aufgaben aus dem Playbook `playbook.yml`, und setzen Sie die folgenden Zeilen an ihre Stelle, um die Aufgabendatei `tasks/web_tasks.yml` zu importieren.

```
- name: Import the web_tasks.yml task file
  import_tasks: tasks/web_tasks.yml
```

3. Strukturieren Sie das Playbook so um, dass die vierte, fünfte und sechste Aufgabe im Playbook in einer externen Aufgabendatei unter `tasks/firewall_tasks.yml` gespeichert werden. Verwenden Sie `import_tasks`, um diese Aufgabendatei in das Playbook zu integrieren.
- 3.1. Platzieren Sie den Inhalt der drei verbleibenden Aufgaben im Playbook `playbook.yml` in der Datei `tasks/firewall_tasks.yml`. Die Aufgabendatei sollte den folgenden Inhalt haben.

```
---
- name: Install firewalld
  yum:
    name: firewalld
    state: latest

- name: Enable and start the firewall
  service:
    name: firewalld
    enabled: true
    state: started

- name: Open the port for http
  firewalld:
    service: http
    immediate: true
    permanent: true
    state: enabled
```

- 3.2. Entfernen Sie die drei verbleibenden Aufgaben aus dem Playbook `playbook.yml`, und setzen Sie die folgenden Zeilen an ihre Stelle, um die Aufgabendatei `tasks/firewall_tasks.yml` zu importieren.

```
- name: Import the firewall_tasks.yml task file
  import_tasks: tasks/firewall_tasks.yml
```

4. Zwischen den Dateien `tasks/web_tasks.yml` und `tasks/firewall_tasks.yml` gibt es einige doppelte Aufgaben. Verschieben Sie die Aufgaben, mit denen Pakete installiert und Services aktiviert werden, in eine neue Datei mit dem Namen `tasks/install_and_enable.yml`, und aktualisieren Sie sie, um Variablen zu verwenden.

**Kapitel 6 |** Verwalten komplexer Plays und Playbooks

Ersetzen Sie die ursprünglichen Aufgaben durch `import_tasks`-Anweisungen, die entsprechende Variablenwerte übergeben.

- 4.1. Kopieren Sie die Aufgaben „yum“ und „service“ aus `tasks/web_tasks.yml` in eine neue Datei namens `tasks/install_and_enable.yml`.

```
---
```

```
- name: Install httpd
  yum:
    name: httpd
    state: latest

- name: Enable and start httpd
  service:
    name: httpd
    enabled: true
    state: started
```

- 4.2. Ersetzen Sie die Paket- und Servicenamen in `tasks/install_and_enable.yml` durch die Variablen `package` und `service`.

```
---
```

```
- name: Install {{ package }}
  yum:
    name: "{{ package }}"
    state: latest

- name: Enable and start {{ service }}
  service:
    name: "{{ service }}"
    enabled: true
    state: started
```

- 4.3. Ersetzen Sie die Aufgaben „yum“ und „service“ in `tasks/web_tasks.yml` und `tasks/firewall_tasks.yml` durch `import_tasks`-Anweisungen.

```
---
```

```
- name: Install and start httpd
  import_tasks: install_and_enable.yml
  vars:
    package: httpd
    service: httpd
```

```
---
```

```
- name: Install and start firewalld
  import_tasks: install_and_enable.yml
  vars:
    package: firewalld
    service: firewalld
```

5. Verifizieren Sie, dass die Änderungen am Playbook `playbook.yml` korrekt vorgenommen wurden, und führen Sie dann das Playbook aus.

- 5.1. Verifizieren Sie, dass das Playbook `playbook.yml` den folgenden Inhalt aufweist.

```
---
- name: Install and configure web service
  hosts: server*.lab.example.com

  tasks:
    - name: Import the web_tasks.yml task file
      import_tasks: tasks/web_tasks.yml

    - name: Import the firewall_tasks.yml task file
      import_tasks: tasks/firewall_tasks.yml

  handlers:
    - name: restart httpd
      service:
        name: httpd
        state: restarted
```

- 5.2. Führen Sie das Playbook mit `ansible-playbook --syntax-check` aus, um zu verifizieren, dass das Playbook keine Syntaxfehler enthält. Wenn Fehler vorhanden sind, korrigieren Sie diese, bevor Sie fortfahren.

```
[student@workstation projects-review]$ ansible-playbook playbook.yml \
> --syntax-check

playbook: playbook.yml
```

- 5.3. Führen Sie das Playbook aus.

```
[student@workstation projects-review]$ ansible-playbook playbook.yml

PLAY [Install and configure web service] ****
TASK [Gathering Facts] ****
ok: [serverd.lab.example.com]
ok: [serverc.lab.example.com]
ok: [serverb.lab.example.com]
ok: [servera.lab.example.com]

TASK [Install httpd] ****
changed: [serverb.lab.example.com]
changed: [servera.lab.example.com]
changed: [serverd.lab.example.com]
changed: [serverc.lab.example.com]

TASK [Enable and start httpd] ****
changed: [servera.lab.example.com]
changed: [serverb.lab.example.com]
changed: [serverd.lab.example.com]
changed: [serverc.lab.example.com]

TASK [Tuning configuration installed] ****
```

```
changed: [serverd.lab.example.com]
changed: [serverc.lab.example.com]
changed: [serverb.lab.example.com]
changed: [servera.lab.example.com]

TASK [Install firewalld] ****
ok: [serverb.lab.example.com]
ok: [servera.lab.example.com]
ok: [serverd.lab.example.com]
ok: [serverc.lab.example.com]

TASK [Enable and start firewalld] ****
ok: [servera.lab.example.com]
ok: [serverb.lab.example.com]
ok: [serverc.lab.example.com]
ok: [serverd.lab.example.com]

TASK [Open the port for http] ****
changed: [serverd.lab.example.com]
changed: [serverb.lab.example.com]
changed: [servera.lab.example.com]
changed: [serverc.lab.example.com]

RUNNING HANDLER [restart httpd] ****
changed: [serverd.lab.example.com]
changed: [serverb.lab.example.com]
changed: [serverc.lab.example.com]
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com    : ok=8    changed=5    unreachable=0    failed=0
serverb.lab.example.com    : ok=8    changed=5    unreachable=0    failed=0
serverc.lab.example.com    : ok=8    changed=5    unreachable=0    failed=0
serverd.lab.example.com    : ok=8    changed=5    unreachable=0    failed=0
```

## Bewertung

Führen Sie auf `workstation` den Befehl `lab projects-review grade` aus, um den Erfolg dieser Übung zu bestätigen. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie das Skript so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab projects-review grade
```

## Beenden

Führen Sie auf `workstation` das Skript `lab projects-review finish` aus, um die in dieser praktischen Übung erstellten Ressourcen zu bereinigen.

```
[student@workstation ~]$ lab projects-review finish
```

Hiermit ist die praktische Übung abgeschlossen.

# Zusammenfassung

---

In diesem Kapitel wurden die folgenden Themen behandelt:

- Mit Host-Pattern werden verwaltete Hosts angegeben, die Ziel eines Plays oder Ad-hoc-Befehls sein sollen.
- Mit dynamischen Inventarskripten können dynamische Listen verwalteter Hosts aus Directory Services oder anderen Quellen außerhalb von Ansible erstellt werden.
- Mit dem Parameter `forks` in der Ansible-Konfigurationsdatei wird die maximale Anzahl paralleler Verbindungen zu verwalteten Hosts festgelegt.
- Mit dem Parameter `serial` können laufende Aktualisierungen für verwaltete Hosts implementiert werden, indem die Anzahl verwalteter Hosts in jedem laufenden Aktualisierungsstapel festgelegt wird.
- Sie können mit `import_playbook` externe Play-Dateien in Playbooks integrieren.
- Sie können mit `include_tasks` oder `import_tasks` externe Play-Dateien in Playbooks integrieren.



## Kapitel 7

# Playbooks mit Rollen vereinfachen

### Ziel

Ansible-Rollen verwenden, um Playbooks schneller zu entwickeln und Ansible-Codes wiederzuverwenden.

### Ziele

- Beschreiben, was eine Rolle ist, wie sie aufgebaut ist und wie sie in einem Playbook verwendet werden kann.
- Playbooks schreiben, die Red Hat Enterprise Linux System Roles zum Ausführen von Standardvorgängen nutzen.
- Erstellen einer Rolle im Projektverzeichnis eines Playbooks und diese als Teil eines Plays im Playbook ausführen.
- Auswählen und Abrufen von Rollen aus der Ansible-Gallery oder anderen Quellen, z. B. einem Git-Repository, und Verwenden in Ihren Playbooks.
- Abrufen einer Reihe von verwandten Rollen, Zusatzmodulen und anderen Inhalten aus Content-Sammlungen und Verwenden in einem Playbook.

### Abschnitte

- Erläutern der Rollenstruktur (und Test)
- Wiederverwenden von Inhalten mit Systemrollen (und angeleitete Übung)
- Erstellen von Rollen (und angeleitete Übung)
- Bereitstellen von Rollen mit Ansible Galaxy (und angeleitete Übung)
- Abrufen von Rollen und Modulen aus Content-Sammlungen (und angeleitete Übung)

### Praktische Übung

- Vereinfachen von Playbooks mit Rollen

# Erläutern der Rollenstruktur

## Ziele

Am Ende dieses Abschnitts sollten Sie zu Folgendem in der Lage sein: Erläutern, was eine Rolle ist, wie sie strukturiert ist und wie sie in einem Playbook verwendet werden kann.

## Strukturieren von Ansible-Playbooks mit Rollen

Wenn Sie mehr Playbooks entwickeln, werden Sie wahrscheinlich feststellen, dass Sie viele Möglichkeiten haben, Code aus bereits erstellten Playbooks wiederzuverwenden. Möglicherweise kann ein Play zur Konfiguration einer MySQL-Datenbank für eine Anwendung mit unterschiedlichen Hostnamen, Passwörtern und Benutzern umgestaltet werden, um eine MySQL-Datenbank für eine andere Anwendung zu konfigurieren.

In der realen Welt kann dieses Play jedoch langwierig und komplex sein, mit vielen eingebundenen oder importierten Dateien sowie mit Aufgaben und Handlern zum Verwalten verschiedener Situationen. Das Kopieren des gesamten Codes in ein anderes Playbook ist möglicherweise auch kein leichtes Unterfangen.

Ansible-Rollen bieten Ihnen eine Möglichkeit, die Wiederverwendung von Ansible-Code allgemein zu vereinfachen. Sie können in eine standardisierte Verzeichnisstruktur alle Aufgaben, Variablen, Dateien, Vorlagen und anderen Ressourcen aufnehmen, die zur Bereitstellung der Infrastruktur oder zum Bereitstellen von Anwendungen erforderlich sind. Sie können diese Rolle von Projekt zu Projekt kopieren, indem Sie einfach das Verzeichnis kopieren. Sie können diese Rolle dann einfach aus einem Play aufrufen, um sie auszuführen.

Eine gut geschriebene Rolle ermöglicht die Übergabe von Variablen aus dem Playbook an die Rolle, die ihr Verhalten anpassen, indem alle standortspezifischen Hostnamen, IP-Adressen, Benutzernamen, Geheimnisse oder anderen ortsspezifischen Details, die Sie benötigen, festgelegt werden. Beispielsweise könnte eine Rolle zum Bereitstellen eines Datenbankservers so geschrieben worden sein, dass sie Variablen unterstützt, mit denen der Hostname, der Benutzer, der Datenbankadministration und das Passwort sowie andere Parameter festgelegt werden, die für Ihre Installation angepasst werden müssen. Der Autor der Rolle kann auch sicherstellen, dass sinnvolle Standardwerte für diese Variablen festgelegt werden, falls Sie sie im Play nicht festlegen.

Ansible-Rollen bieten die folgenden Vorteile:

- Rollengruppeninhalt, der eine einfache Teilung des Codes ermöglicht
- Rollen können geschrieben werden, die die wichtigen Elemente eines Systemtyps definieren: Webserver, Datenbankserver, Git-Repository oder für andere Zwecke
- Rollen machen größere Projekte übersichtlicher
- Rollen können parallel von unterschiedlichen Administratoren entwickelt werden

Sie können nicht nur eigene Rollen schreiben, verwenden, wiederverwenden und freigeben, sondern auch Rollen von anderen Quellen. Einige Rollen sind als Bestandteil von Red Hat Enterprise Linux im *rhel-system-roles*-Paket enthalten. Sie erhalten auch zahlreiche von der Community unterstützte Rollen auf der Ansible Galaxy-Website. Später in diesem Kapitel erfahren Sie mehr über diese Rollen.

## Untersuchen der Ansible-Rollenstruktur

Eine Ansible-Rolle wird durch eine standardisierte Struktur von Unterverzeichnissen und Dateien definiert. Das oberste Verzeichnis definiert den Namen der Rolle. Dateien sind in Unterverzeichnissen organisiert, die nach dem Zweck der jeweiligen Datei in der Rolle benannt sind, wie z. B. `tasks` und `handlers`. Die Unterverzeichnisse `files` und `templates` enthalten Dateien, auf die von Aufgaben in anderen YAML-Dateien verwiesen wird.

Der folgende Befehl `tree` zeigt die Verzeichnisstruktur der Rolle `user.example` an.

```
[user@host roles]$ tree user.example
user.example/
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml
```

### Ansible Rollenunterverzeichnisse

| Unterverzeichnis       | Funktion                                                                                                                                                                                                                                             |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>defaults</code>  | Die Datei <code>main.yml</code> in diesem Verzeichnis enthält die Standardwerte der Rollenvariablen, die bei Verwendung überschrieben werden können. Diese Variablen haben eine geringe Priorität und sollen in Plays geändert und angepasst werden. |
| <code>files</code>     | Dieses Verzeichnis enthält statische Dateien, auf die Rollenaufgaben verweisen.                                                                                                                                                                      |
| <code>handlers</code>  | Die Datei „ <code>main.yml</code> “ in diesem Verzeichnis enthält die Handler-Definitionen der Rolle.                                                                                                                                                |
| <code>meta</code>      | Die Datei <code>main.yml</code> in diesem Verzeichnis enthält Informationen über die Rolle, inklusive Autor, Lizenz, Plattformen und optionale Rollenabhängigkeiten.                                                                                 |
| <code>tasks</code>     | Die Datei „ <code>main.yml</code> “ in diesem Verzeichnis enthält die Aufgabendefinitionen der Rolle.                                                                                                                                                |
| <code>templates</code> | Dieses Verzeichnis enthält Jinja2-Vorlagen, auf die die Rollenaufgaben verweisen.                                                                                                                                                                    |

| Unterverzeichnis | Funktion                                                                                                                                                                                                                                                                        |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| tests            | Dieses Verzeichnis kann ein Inventar und Playbook test.yml enthalten, die für die Prüfung der Rolle verwendet werden können.                                                                                                                                                    |
| vars             | Die Datei „main.yml“ in diesem Verzeichnis definiert die Variablenwerte der Rolle. Diese Variablen werden häufig für interne Zwecke innerhalb der Rolle verwendet. Diese Variablen haben eine hohe Priorität und sollen bei Verwendung in einem Playbook nicht geändert werden. |

Nicht jede Rolle verfügt über alle diese Verzeichnisse.

## Definieren von Variablen und Standardwerten

*Rollenvariablen* werden durch die Erstellung einer vars/main.yml-Datei mit Schlüssel:Wert-Paaren in der Rollenverzeichnishierarchie definiert. Sie werden in der Rollen-YAML-Datei wie jede andere Variable angegeben: {{ VAR\_NAME }}. Diese Variablen haben eine hohe Priorität und können nicht von Inventarvariablen überschrieben werden. Der Zweck dieser Variablen ist deren Verwendung zum internen Funktionieren der Rolle.

*Standardvariablen* ermöglichen, dass Standardwerte für Variablen festgelegt werden, die in einem Play zum Konfigurieren der Rolle oder zum Anpassen ihres Verhaltens verwendet werden können. Sie werden durch die Erstellung einer defaults/main.yml-Datei mit Schlüssel:Wert-Paaren in der Rollenverzeichnishierarchie definiert. Standardvariablen haben die niedrigste Priorität aller verfügbaren Variablen. Sie können leicht von anderen Variablen, inklusive der Inventarvariablen, überschrieben werden. Diese Variablen sollen es der Person, die ein Play schreibt, in dem die Rolle verwendet wird, ermöglichen, das, was die Rolle ausführt, zu ändern oder genau zu steuern. Sie können verwendet werden, um Informationen zu der Rolle bereitzustellen, die zum ordnungsgemäßen Konfigurieren oder Bereitstellen von etwas benötigt werden.

Definieren Sie eine spezifische Variable entweder in vars/main.yml oder defaults/main.yml, aber nicht an beiden Orten. Standardvariablen sollten dann genutzt werden, wenn eine Überschreibung ihrer Werte geplant ist.



### Wichtig

Rollen sollten keine standortspezifischen Daten enthalten. Sie sollten auf keinen Fall Geheimnisse wie Passwörter oder private Schlüssel enthalten.

Und zwar, weil Rollen allgemein, wiederverwendbar und frei gemeinsam nutzbar sein sollen. Standortspezifische Details sollten in Rollen nicht hart kodiert sein.

Secrets sollten der Rolle auf andere Weise zur Verfügung gestellt werden. Dies ist ein Grund, warum Sie beim Aufrufen einer Rolle Rollenvariablen festlegen sollten. Im Play festgelegte Rollenvariablen können das Geheimnis bereitstellen oder auf eine mit Ansible Vault verschlüsselte Datei verweisen, die das Geheimnis enthält.

## Verwenden von Ansible-Rollen in einem Playbook

Die Verwendung von Rollen in einem Playbook ist sehr einfach. Das folgende Beispiel zeigt eine Möglichkeit der Verwendung von Ansible-Rollen.

```
---
```

```
- hosts: remote.example.com
  roles:
    - role1
    - role2
```

Für jede angegebene Rolle werden die Rollenaufgaben, Rollen-Handler, Rollenvariablen und Rollenabhängigkeiten im Playbook in dieser Reihenfolge importiert. Jede der Aufgaben `copy`, `script`, `template` oder `include_tasks/import_tasks` in der Rolle kann auf die relevanten Dateien, Vorlagen oder Aufgabendateien in der Rolle ohne absolute oder relative Pfadnamen verweisen. Ansible sucht sie in den Unterverzeichnissen `files`, `templates` bzw. `tasks` der Rolle.

Wenn Sie einen `roles`-Abschnitt verwenden, um Rollen in ein Play zu importieren, werden die Rollen vor den von Ihnen definierten Aufgaben für dieses Play ausgeführt.

Im folgenden Beispiel werden Werte für zwei Rollenvariablen von `role2`, `var1` und `var2` festgelegt. Alle `defaults`- und `vars`-Variablen werden überschrieben, wenn `role2` verwendet wird.

```
---
```

```
- hosts: remote.example.com
  roles:
    - role: role1
    - role: role2
      var1: val1
      var2: val2
```

Eine andere äquivalente YAML-Syntax für diesen Fall wäre:

```
---
```

```
- hosts: remote.example.com
  roles:
    - role: role1
    - { role: role2, var1: val1, var2: val2 }
```

Es gibt Situationen, in denen dies schwieriger zu lesen ist, obwohl es kompakter ist.



### Wichtig

Inline festgelegte Rollenvariablen (Rollenparameter), wie in den vorherigen Beispielen, haben eine sehr hohe Priorität. Sie überschreiben die meisten anderen Variablen.

Achten Sie darauf, die Namen von Rollenvariablen, die Sie an anderer Stelle in Ihrem Play festgelegt haben, nicht erneut zu verwenden, da die Werte der Rollenvariablen Inventarvariablen und alle anderen Play-`vars` überschreiben.

## Steuern der Ausführungsreihenfolge

Für jedes Play in einem Playbook werden Aufgaben wie in der Aufgabenliste geordnet ausgeführt. Nachdem alle Aufgaben ausgeführt wurden, werden alle benachrichtigten Handler ausgeführt.

## Kapitel 7 | Playbooks mit Rollen vereinfachen

Wenn einem Play eine Rolle hinzugefügt wird, werden Rollenaufgaben am Anfang der Aufgabenliste hinzugefügt. Wenn eine zweite Rolle in ein Play einbezogen wird, wird ihre Aufgabenliste nach der ersten Rolle hinzugefügt.

Rollen-Handler werden Plays auf dieselbe Weise hinzugefügt wie Rollenaufgaben zu Plays. Jedes Play definiert eine Liste mit Handlern. Rollen-Handler werden zuerst der Liste der Handler hinzugefügt, gefolgt von den im Abschnitt `handler` des Plays definierten Handlern.

In bestimmten Szenarien müssen möglicherweise einige Play-Aufgaben vor den Rollen ausgeführt werden. Um solche Szenarien zu unterstützen, können Plays mit einem `pre_tasks`-Abschnitt konfiguriert werden. Alle in diesem Abschnitt aufgeführten Aufgaben werden vor den Rollen ausgeführt. Wenn eine dieser Aufgaben einen Handler benachrichtigt, werden diese Handler-Aufgaben vor den Rollen oder normalen Aufgaben ausgeführt.

Plays unterstützen auch das Keyword `post_tasks`. Diese Aufgaben werden nach den normalen Aufgaben des Plays ausgeführt und alle Handler, die sie benachrichtigen, werden ausgeführt.

Das folgende Play zeigt ein Beispiel mit `pre_tasks`, `roles`, `tasks`, `post_tasks` und `handlers`. Es wäre ungewöhnlich, dass ein Play alle diese Abschnitte enthält.

```
- name: Play to illustrate order of execution
  hosts: remote.example.com
  pre_tasks:
    - debug:
        msg: 'pre-task'
        notify: my handler
  roles:
    - role1
  tasks:
    - debug:
        msg: 'first task'
        notify: my handler
  post_tasks:
    - debug:
        msg: 'post-task'
        notify: my handler
  handlers:
    - name: my handler
      debug:
        msg: Running my handler
```

Im obigen Beispiel wird in jedem Abschnitt eine `debug`-Aufgabe ausgeführt, um den Handler `my handler` zu benachrichtigen. Die Aufgabe `my handler` wird dreimal ausgeführt:

- nach der Ausführung aller `pre_tasks`-Aufgaben
- nach der Ausführung aller Rollenaufgaben und Aufgaben aus dem Abschnitt `tasks`
- nach der Ausführung aller `post_tasks`

Rollen können Plays mit einer normalen Aufgabe hinzugefügt werden, nicht nur durch das Einfügen in den Abschnitt `roles` eines Plays. Verwenden Sie das Modul `include_role`, um dynamisch eine Rolle einzubeziehen, und verwenden Sie das Modul `import_role` zum statischen Importieren einer Rolle.

## Kapitel 7 | Playbooks mit Rollen vereinfachen

Das folgende Playbook veranschaulicht, wie eine Rolle mithilfe einer Aufgabe mit dem Modul `include_role` einbezogen werden kann.

```
- name: Execute a role as a task
  hosts: remote.example.com
  tasks:
    - name: A normal task
      debug:
        msg: 'first task'
    - name: A task to include role2 here
      include_role: role2
```



### Anmerkung

Das Modul `include_role` wurde in Ansible 2.3 hinzugefügt und das Modul `import_role` in Ansible 2.4.



### Literaturhinweise

#### Roles – Ansible-Dokumentation

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_reuse\\_roles.html](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_reuse_roles.html)

## ► Quiz

# Erläutern der Rollenstruktur

Wählen Sie die richtige Antwort auf die folgenden Fragen aus:

Klicken Sie nach Abschluss des Tests auf **check**. Wenn Sie es noch einmal versuchen möchten, klicken Sie auf **reset**. Klicken Sie auf **show solution**, um alle korrekten Antworten anzuzeigen.

### ► 1. Welche der folgenden Aussagen beschreibt Rollen am besten?

- a. Konfigurationseinstellungen, die es spezifischen Benutzern ermöglichen, Ansible-Playbooks auszuführen.
- b. Playbooks für ein Rechenzentrum.
- c. Sammlung von YAML-Aufgabendateien und unterstützende Objekte, die in einer spezifischen Struktur zur einfachen Teilung, Portabilität und Wiederverwendung organisiert sind.

### ► 2. Welche der folgenden Elemente können in Rollen spezifiziert werden?

- a. Handler
- b. Aufgaben
- c. Vorlagen
- d. Variablen
- e. Alle oben genannten

### ► 3. Welche Datei deklariert Rollenabhängigkeiten?

- a. Das Ansible-Playbook, das die Rolle verwendet.
- b. Die Datei `meta/main.yml` innerhalb der Rollenhierarchie.
- c. Die Datei `meta/main.yml` im Projektverzeichnis.
- d. Rollenabhängigkeiten können in Ansible nicht definiert werden.

### ► 4. Welche Datei in einer Rollenverzeichnishierarchie sollte die Anfangswerte von Variablen enthalten, die ggf. als Parameter der Rolle verwendet werden?

- a. `default/main.yml`
- b. `meta/main.yml`
- c. `vars/main.yml`
- d. Die Inventardatei des Hosts.

## ► Lösung

# Erläutern der Rollenstruktur

Wählen Sie die richtige Antwort auf die folgenden Fragen aus:

Klicken Sie nach Abschluss des Tests auf **check**. Wenn Sie es noch einmal versuchen möchten, klicken Sie auf **reset**. Klicken Sie auf **show solution**, um alle korrekten Antworten anzuzeigen.

► 1. Welche der folgenden Aussagen beschreibt Rollen am besten?

- a. Konfigurationseinstellungen, die es spezifischen Benutzern ermöglichen, Ansible-Playbooks auszuführen.
- b. Playbooks für ein Rechenzentrum.
- c. Sammlung von YAML-Aufgabendateien und unterstützende Objekte, die in einer spezifischen Struktur zur einfachen Teilung, Portabilität und Wiederverwendung organisiert sind.

► 2. Welche der folgenden Elemente können in Rollen spezifiziert werden?

- a. Handler
- b. Aufgaben
- c. Vorlagen
- d. Variablen
- e. Alle oben genannten

► 3. Welche Datei deklariert Rollenabhängigkeiten?

- a. Das Ansible-Playbook, das die Rolle verwendet.
- b. Die Datei `meta/main.yml` innerhalb der Rollenhierarchie.
- c. Die Datei `meta/main.yml` im Projektverzeichnis.
- d. Rollenabhängigkeiten können in Ansible nicht definiert werden.

► 4. Welche Datei in einer Rollenverzeichnishierarchie sollte die Anfangswerte von Variablen enthalten, die ggf. als Parameter der Rolle verwendet werden?

- a. `default/main.yml`
- b. `meta/main.yml`
- c. `vars/main.yml`
- d. Die Inventardatei des Hosts.

# Wiederverwenden von Inhalten mit Systemrollen

---

## Ziele

Am Ende dieses Abschnitts sollten Sie in der Lage sein, Playbooks zu schreiben, die Red Hat Enterprise Linux System Roles zum Ausführen von Standardvorgängen nutzen.

## Red Hat Enterprise Linux System Roles

Ab Red Hat Enterprise Linux 7.4 wurde eine Reihe von Ansible-Rollen mit dem Betriebssystem im Paket *rhel-system-roles* bereitgestellt. Für Red Hat Enterprise Linux 8 ist das Paket im AppStream-Channel verfügbar. Kurzbeschreibung der Rollen:

### RHEL-Systemrollen

| Name                                    | Status                  | Rollenbeschreibung                                                                                                                                      |
|-----------------------------------------|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>rhel-system-roles.kdump</code>    | Vollständig unterstützt | Konfiguriert den kdump-Service zur Wiederherstellung nach Abstürzen.                                                                                    |
| <code>rhel-system-roles.network</code>  | Vollständig unterstützt | Konfiguriert Netzwerkschnittstellen.                                                                                                                    |
| <code>rhel-system-roles.selinux</code>  | Vollständig unterstützt | Konfiguriert und verwaltet die SELinux-Anpassung, einschließlich SELinux-Modus, Datei- und Portkontexte, boolescher Einstellungen und SELinux-Benutzer. |
| <code>rhel-system-roles.timesync</code> | Vollständig unterstützt | Konfiguriert die Zeitsynchronisierung mithilfe des Network Time Protocol oder Precision Time Protocol.                                                  |
| <code>rhel-system-roles.postfix</code>  | Technologievorschau     | Konfiguriert mit dem Postfix-Service jeden Host als Mail Transfer Agent.                                                                                |
| <code>rhel-system-roles.firewall</code> | In Entwicklung          | Konfiguriert die Firewall eines Hosts.                                                                                                                  |
| <code>rhel-system-roles.tuned</code>    | In Entwicklung          | Konfiguriert den Service <code>tuned</code> zur Optimierung der Systemleistung.                                                                         |

Mit Systemrollen wird die Konfiguration von Red Hat Enterprise Linux-Subsystemen für mehrere Versionen standardisiert. Verwenden Sie Systemrollen, um Red Hat Enterprise Linux ab Version 6.10 zu konfigurieren.

## Vereinfachte Konfigurationsverwaltung

Zum Beispiel ist der empfohlene Zeitsynchronisierungsservice für Red Hat Enterprise Linux 7 der Service `chrony`. In Red Hat Enterprise Linux 6 ist der empfohlene Service dagegen `ntpd`. In einer Umgebung mit einer Kombination aus Red Hat Enterprise Linux 6- und 7-Hosts muss der Administrator die Konfigurationsdateien für beide Services verwalten.

Mit RHEL System Roles müssen Administratoren keine Konfigurationsdateien mehr für beide Services verwalten. Administratoren können mit der Rolle `rhel-system-roles.timesync` die Zeitsynchronisierung für Hosts von beiden Versionen, Red Hat Enterprise Linux 6 und 7, konfigurieren. In einer vereinfachten YAML-Datei mit Rollenvariablen ist die Konfiguration der Zeitsynchronisierung für beide Hosttypen definiert.

## Unterstützung für RHEL System Roles

RHEL System Roles sind vom Open Source-Projekt „Linux System Roles“ abgeleitet, das sich auf Ansible Galaxy befindet. Im Gegensatz zu Linux System Roles werden RHEL System Roles von Red Hat im Rahmen eines Standardabonnements von Red Hat Enterprise Linux unterstützt. RHEL System Roles bieten dieselben Vorteile für den Lebenszyklus wie ein Red Hat Enterprise Linux-Abonnement.

Alle Systemrollen wurden getestet und sind stabil. Die `vollständig unterstützten` Systemrollen verfügen auch über stabile Schnittstellen. Für jede Systemrolle des Typs `Vollständig unterstützt` versucht Red Hat sicherzustellen, dass die Rollenvariablen in künftigen Versionen unverändert bleiben. Das Refactoring von Playbooks aufgrund von Systemrollenänderungen sollte minimal sein.

Die `Technologievorschau`-Systemrollen können in zukünftigen Versionen andere Rollenvariablen verwenden. Für Playbooks, die `Technologievorschau`-Rollen enthalten, werden Integrationstests empfohlen. Für Playbooks muss möglicherweise ein Refactoring vorgenommen werden, wenn sich Rollenvariablen in einer zukünftigen Version der Rolle ändern.

Weitere Rollen befinden sich in der Entwicklung des Upstream-Projekts „Linux System Roles“, sind jedoch noch nicht im Rahmen einer RHEL-Subskription verfügbar. Diese Rollen sind über Ansible Galaxy verfügbar.

## Installieren von RHEL System Roles

Die RHEL System Roles werden im `rhel-system-roles`-Paket bereitgestellt, das im AppStream-Channel verfügbar ist. Installieren Sie dieses Paket auf dem Ansible-Kontrollknoten.

Gehen Sie folgendermaßen vor, um das `rhel-system-roles`-Paket zu installieren. Bei diesem Vorgehen wird vorausgesetzt, dass der Kontrollknoten beim Red Hat Enterprise Linux-Abonnement registriert und Ansible installiert ist. Weitere Informationen finden Sie im Abschnitt *Installieren von Ansible*.

1. Installieren Sie RHEL System Roles.

```
[root@host ~]# yum install rhel-system-roles
```

Nach der Installation befinden sich die RHEL System Roles im Verzeichnis `/usr/share/ansible/roles`:

```
[root@host ~]# ls -l /usr/share/ansible/roles/
total 20
...output omitted... linux-system-roles.kdump -> rhel-system-roles.kdump
```

**Kapitel 7 |** Playbooks mit Rollen vereinfachen

```
...output omitted... linux-system-roles.network -> rhel-system-roles.network
...output omitted... linux-system-roles.postfix -> rhel-system-roles.postfix
...output omitted... linux-system-roles.selinux -> rhel-system-roles.selinux
...output omitted... linux-system-roles.timesync -> rhel-system-roles.timesync
...output omitted... rhel-system-roles.kdump
...output omitted... rhel-system-roles.network
...output omitted... rhel-system-roles.postfix
...output omitted... rhel-system-roles.selinux
...output omitted... rhel-system-roles.timesync
```

Der entsprechende Upstream-Name jeder Rolle ist mit der RHEL System Role verknüpft. Daher kann auf eine Rolle mit einem der beiden Namen in einem Playbook verwiesen werden.

Der Standardpfad `roles_path` zu Red Hat Enterprise Linux enthält im Pfad `/usr/share/ansible/roles`, daher sollte Ansible diese Rollen automatisch finden, wenn in einem Playbook auf sie verwiesen wird.

**Anmerkung**

Ansible findet die Systemrollen möglicherweise nicht, wenn `roles_path` in der aktuellen Ansible-Konfigurationsdatei überschrieben wurde, wenn die Umgebungsvariable `ANSIBLE_ROLES_PATH` festgelegt ist oder wenn eine andere Rolle mit demselben Namen in einem zuvor in `roles_path` aufgeführten Verzeichnis vorhanden ist.

## Öffnen der Dokumentation zu RHEL System Roles

Nach der Installation befindet sich die Dokumentation für RHEL System Roles im Verzeichnis `/usr/share/doc/rhel-system-roles-<version>/`. Die Dokumentation ist nach Subsystem in Unterverzeichnisse aufgeteilt:

```
[root@host ~]# ls -l /usr/share/doc/rhel-system-roles/
total 4
drwxr-xr-x. ...output omitted... kdump
drwxr-xr-x. ...output omitted... network
drwxr-xr-x. ...output omitted... postfix
drwxr-xr-x. ...output omitted... selinux
drwxr-xr-x. ...output omitted... timesync
```

Das Dokumentationsverzeichnis jeder Rolle enthält eine `README.md`-Datei. Die `README.md`-Datei enthält eine Beschreibung der Rolle sowie Informationen zu deren Verwendung.

In der `README.md`-Datei sind auch Rollenvariablen beschrieben, die das Verhalten der Rolle beeinflussen. Einige `README.md`-Dateien enthalten ein Playbook-Codefragment, das die Variableneinstellungen für ein allgemeines Konfigurationsszenario veranschaulicht.

In einigen Verzeichnissen der Rollendokumentationen sind Beispiel-Playbooks enthalten. Überprüfen Sie bei der erstmaligen Verwendung einer Rolle alle zusätzlichen Beispiel-Playbooks im Dokumentationsverzeichnis.

Die Rollendokumentation für RHEL System Roles entspricht der Dokumentation für Linux System Roles. Verwenden Sie zum Anzeigen der Rollendokumentation für die Upstream-Rollen auf der Ansible Galaxy-Website <https://galaxy.ansible.com> einen Webbrowser.

## Beispiel für die Zeitsynchronisierungsrolle

Angenommen, Sie müssen die NTP-Zeitsynchronisierung auf Ihren Servern konfigurieren. Sie könnten selbst eine Automatisierung schreiben, um alle erforderlichen Aufgaben auszuführen. RHEL System Roles beinhaltet jedoch eine Rolle, die dies für Sie erledigt, `rhel-system-roles.timesync`.

Die Rolle ist in der Datei `README.md` im Verzeichnis `/usr/share/doc/rhel-system-roles/timesync` dokumentiert. In der Datei sind alle Variablen beschrieben, die das Verhalten der Rolle beeinflussen. Außerdem sind drei Playbook-Codefragmente enthalten, die verschiedene Konfigurationen für die Zeitsynchronisierung veranschaulichen.

Für die manuelle Konfiguration von NTP-Servern verfügt die Rolle über die Variable `timesync_ntp_servers`. Sie übernimmt eine Liste der zu verwendenden NTP-Server. Jedes Element auf der Liste besteht aus einem oder mehreren Attributen. Die beiden Schlüsselattribute sind:

### **timesync\_ntp\_servers-Attribute**

| Attribute           | Zweck                                                                                                                                                                                                                    |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Hostname            | Der Hostname eines NTP-Servers, mit dem synchronisiert werden soll.                                                                                                                                                      |
| <code>iburst</code> | Ein boolescher Wert, der die schnelle Erstsynchronisierung aktiviert oder deaktiviert. Standardeinstellung in der Rolle ist <code>no</code> . In der Regel sollten Sie diese Einstellung auf <code>yes</code> festlegen. |

Ausgehend von diesen Informationen ist das folgende Beispiel ein Play, in dem mit der Rolle `rhel-system-roles.timesync` verwaltete Hosts konfiguriert werden, um die Zeit anhand der schnellen Erstsynchronisierung von drei NTP-Servern abzurufen. Außerdem wurde eine Aufgabe hinzugefügt, die mit dem Modul `timezone` die Zeitzone der Hosts auf UTC einstellt.

```
- name: Time Synchronization Play
hosts: servers
vars:
  timesync_ntp_servers:
    - hostname: 0.rhel.pool.ntp.org
      iburst: yes
    - hostname: 1.rhel.pool.ntp.org
      iburst: yes
    - hostname: 2.rhel.pool.ntp.org
      iburst: yes
  timezone: UTC

roles:
  - rhel-system-roles.timesync

tasks:
  - name: Set timezone
    timezone:
      name: "{{ timezone }}"
```



### Anmerkung

Wenn Sie eine andere Zeitzone einstellen möchten, können Sie mit dem Befehl `tzselect` nach anderen gültigen Werten suchen. Sie können außerdem mit dem Befehl `timedatectl` die aktuellen Uhreinstellungen überprüfen.

In diesem Beispiel werden die Rollenvariablen im Abschnitt `vars` des Plays festgelegt, eine bessere Vorgehensweise wäre möglicherweise, sie als Inventarvariablen für Hosts oder Hostgruppen zu konfigurieren.

Ziehen Sie ein Playbook-Projekt mit der folgenden Struktur in Erwägung:

```
[root@host playbook-project]# tree
.
├── ansible.cfg
├── group_vars
│   └── servers
│       └── timesync.yml①
└── inventory
    └── timesync_playbook.yml②
```

- ① Definiert die Zeitsynchronisierungsvariablen, die die Standardeinstellungen der Rolle für Hosts in der Gruppe `servers` im Inventar überschreiben. Diese Datei würde ungefähr so aussehen:

```
timesync_ntp_servers:
  - hostname: 0.rhel.pool.ntp.org
    iburst: yes
  - hostname: 1.rhel.pool.ntp.org
    iburst: yes
  - hostname: 2.rhel.pool.ntp.org
    iburst: yes
timezone: UTC
```

- ② Der Inhalt des Playbooks wird zu Folgendem vereinfacht:

```
- name: Time Synchronization Play
  hosts: servers
  roles:
    - rhel-system-roles.timesync
  tasks:
    - name: Set timezone
      timezone:
        name: "{{ timezone }}"
```

Diese Struktur sorgt für eine eindeutige Trennung der Rolle, des Playbook-Codes und der Konfigurationseinstellungen. Der Playbook-Code ist einfach, leicht lesbar und sollte kein komplexes Refactoring erfordern. Der Rolleninhalt wird von Red Hat verwaltet und unterstützt. Alle Einstellungen werden als Inventarvariablen behandelt.

Diese Struktur unterstützt auch eine dynamische, heterogene Umgebung. Hosts mit neuen Zeitsynchronisierungsanforderungen können in eine neue Hostgruppe aufgenommen

werden. Geeignete Variablen werden in einer YAML-Datei definiert und in das entsprechende Unterverzeichnis `group_vars` (oder `host_vars`) eingefügt.

## Beispiel für eine SELinux-Rolle

Als weiteres Beispiel vereinfacht die Rolle `rhel-system-roles.selinux` die Verwaltung von SELinux-Konfigurationseinstellungen. Sie wird mit den SELinux-bezogenen Ansible-Modulen implementiert. Der Vorteil der Verwendung dieser Rolle ist, dass Sie keine eigenen Aufgaben schreiben müssen. Stattdessen konfigurieren Sie die Rolle mit Variablen und der verwaltete Code in der Rolle stellt sicher, dass Ihre gewünschte SELinux-Konfiguration angewendet wird.

Unter anderem kann diese Rolle folgende Aufgaben ausführen:

- Enforcing- oder Permissive-Modus festlegen
- `restorecon` für Teile der Dateisystemhierarchie ausführen
- Boolesche SELinux-Werte festlegen
- SELinux-Dateikontexte dauerhaft festlegen
- SELinux-Benutzerzuordnungen festlegen

## Aufrufen der SELinux-Rolle

Manchmal muss die SELinux-Rolle dafür sorgen, dass die verwalteten Hosts neu gebootet werden, damit die Änderungen vollständig übernommen werden. Die Rolle führt jedoch niemals selbst einen Reboot der Hosts durch. Damit wird sichergestellt, dass Sie steuern können, wie der Reboot gehandhabt wird. Das bedeutet jedoch, dass die korrekte Verwendung dieser Rolle in einem Play etwas komplizierter als üblich ist.

Das funktioniert so, dass die Rolle die boolesche Variable `selinux_reboot_required` auf `true` festlegt und fehlschlägt, wenn ein Reboot erforderlich ist. Sie können eine `block/rescue`-Struktur für die Wiederherstellung nach einem Fehler verwenden. Das Play schlägt fehl, wenn diese Variable nicht auf `true` festgelegt ist, oder der verwaltete Host wird neu gebootet und die Rolle erneut ausgeführt, wenn die Rolle auf `true` festgelegt ist. Der Block in Ihrem Play sollte ungefähr so aussehen:

```
- name: Apply SELinux role
  block:
    - include_role:
        name: rhel-system-roles.selinux
  rescue:
    - name: Check for failure for other reasons than required reboot
      fail:
        when: not selinux_reboot_required

    - name: Restart managed host
      reboot:

    - name: Reapply SELinux role to complete changes
      include_role:
        name: rhel-system-roles.selinux
```

## Konfigurieren der SELinux-Rolle

Die für die Konfiguration der `rhel-system-roles.selinux`-Rolle verwendeten Variablen sind in der entsprechenden `README.md`-Datei dokumentiert. Die folgenden Beispiele zeigen einige Möglichkeiten für die Verwendung dieser Rolle.

Die Variable `selinux_state` legt den Modus fest, in dem SELinux ausgeführt wird. Sie kann auf `enforcing`, `permissive` oder `disabled` festgelegt werden. Wenn sie nicht festgelegt ist, wird der Modus nicht geändert.

```
selinux_state: enforcing
```

Die Variable `selinux_booleans` übernimmt eine Liste mit booleschen SELinux-Werten zum Anpassen. Jedes Element auf der Liste ist ein Hash-Wert/Wörterbuch von Variablen: der name des booleschen Werts, der state (on oder off) und ob die Einstellung auch nach Reboots persistent sein soll oder nicht.

Dieses Beispiel legt `httpd_enable_homedirs` dauerhaft auf on fest:

```
selinux_booleans:
- name: 'httpd_enable_homedirs'
  state: 'on'
  persistent: 'yes'
```

Die Variable `selinux_fcontext` übernimmt eine Liste mit Dateikontexten, um sie dauerhaft festzulegen (oder zu entfernen). Diese Variable arbeitet ähnlich wie der Befehl `seLinux fcontext`.

Im folgenden Beispiel wird sichergestellt, dass die Richtlinie über eine Regel verfügt, um den SELinux-Standardtyp für alle Dateien unter `/srv/www` auf `httpd_sys_content_t` festzulegen.

```
selinux_fcontexts:
- target: '/srv/www(/.*)?'
  setype: 'httpd_sys_content_t'
  state: 'present'
```

Die Variable `selinux_restore_dirs` gibt eine Liste der Verzeichnisse an, für die `restorecon` ausgeführt werden soll:

```
selinux_restore_dirs:
- /srv/www
```

Die Variable `selinux_ports` übernimmt eine Liste mit Ports, für die ein bestimmter SELinux-Typ festgelegt sein sollte.

```
selinux_ports:
- ports: '82'
  setype: 'http_port_t'
  proto: 'tcp'
  state: 'present'
```

Für diese Rolle stehen noch weitere Variablen und Optionen zur Verfügung. Weitere Informationen finden Sie in der entsprechenden `README.md`-Datei.



### Literaturhinweise

#### Red Hat Enterprise Linux (RHEL) System Roles

<https://access.redhat.com/articles/3050101>

#### Linux-Systemrollen

<https://linux-system-roles.github.io/>

## ► Angeleitete Übung

# Wiederverwenden von Inhalten mit Systemrollen

In dieser Übung verwenden Sie eine der Red Hat Enterprise Linux System Roles in Verbindung mit einer normalen Aufgabe zum Konfigurieren der Zeitsynchronisierung und der Zeitzone auf verwalteten Hosts.

## Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Installieren der Red Hat Enterprise Linux System Roles
- Suchen und Verwenden der Dokumentation zu RHEL System Roles
- Verwenden der Rolle `rhel-system-roles.timesync` in einem Playbook, um die Zeitsynchronisierung auf Remote-Hosts zu konfigurieren

## Übersicht über das Szenario

Ihr Unternehmen unterhält zwei Rechenzentren: eines in den USA (Chicago) und eines in Finnland (Helsinki). Stellen Sie zur Unterstützung der Protokollanalyse von Datenbankservern mit dem Network Time Protocol in allen Rechenzentren sicher, dass die Systemuhr auf allen Hosts synchronisiert ist. Um die Tageszeit-Aktivitätsanalyse in den Rechenzentren zu unterstützen, stellen Sie sicher, dass für jeden Datenbankserver eine dem Standort des Host-Rechenzentrums entsprechende Zeitzone festgelegt ist.

Für die Zeitsynchronisierung gelten folgende Anforderungen:

- Verwenden Sie den NTP-Server unter `classroom.example.com`. Aktivieren Sie die Option `iburst` zur Beschleunigung der ersten Zeitsynchronisierung.
- Verwenden Sie das Paket `chrony` für die Zeitsynchronisierung.

## Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab role-system start` aus. Damit wird das Arbeitsverzeichnis `/home/student/role-system` erstellt und darin eine Ansible-Konfigurationsdatei und das Hostinventar gespeichert.

```
[student@workstation ~]$ lab role-system start
```

## Anweisungen

- 1. Wechseln Sie in das Arbeitsverzeichnis `/home/student/role-system`.

```
[student@workstation ~]$ cd ~/role-system  
[student@workstation role-system]$
```

- 2. Installieren Sie die Red Hat Enterprise Linux System Roles auf dem Kontrollknoten `workstation.lab.example.com`. Verifizieren Sie den Installationsort der Rollen auf dem Kontrollknoten.
- 2.1. Verifizieren Sie mit dem Befehl `ansible-galaxy`, dass anfänglich keine Rollen für das Playbook-Projekt verfügbar sind.

```
[student@workstation role-system]$ ansible-galaxy list  
# /home/student/role-system/roles  
# /usr/share/ansible/roles  
# /etc/ansible/roles
```

Der Befehl `ansible-galaxy` durchsucht entsprechend dem Eintrag `roles_path` in der Datei `ansible.cfg` drei Verzeichnisse nach Rollen:

- `./roles`
- `/usr/share/ansible/roles`
- `/etc/ansible/roles`

Die obige Ausgabe zeigt, dass in keinem dieser Verzeichnisse Rollen vorhanden sind.

- 2.2. Installieren Sie das Paket `rhel-system-roles`.

```
[student@workstation role-system]$ sudo yum install rhel-system-roles
```

Geben Sie `y` ein, wenn Sie aufgefordert werden, das Paket zu installieren.

- 2.3. Verifizieren Sie mit dem Befehl `ansible-galaxy`, dass die Systemrollen jetzt verfügbar sind.

```
[student@workstation role-system]$ ansible-galaxy list  
# /home/student/role-system/roles  
# /usr/share/ansible/roles  
...output omitted...  
- rhel-system-roles.timesync, (unknown version)  
- rhel-system-roles.tlog, (unknown version)  
# /etc/ansible/roles
```

Die Rollen befinden sich im Verzeichnis `/usr/share/ansible/roles`. Jede Rolle, die mit `linux-system-roles` beginnt, ist eigentlich ein Symlink zur entsprechenden `rhel-system-roles`-Rolle.

- 3. Erstellen Sie das Playbook `configure_time.yml` mit einem Play, dessen Ziel die Hostgruppe `database_servers` ist. Nehmen Sie die Rolle `rhel-system-roles.timesync` in den Abschnitt `roles` des Plays auf.

```
---  
- name: Time Synchronization  
  hosts: database_servers  
  
  roles:  
    - rhel-system-roles.timesync
```

**Kapitel 7 |** Playbooks mit Rollen vereinfachen

- 4. Die Rollendokumentation enthält eine Beschreibung aller Rollenvariablen, einschließlich des Standardwerts für jede Variable. Legen Sie die zu überschreibenden Rollenvariablen fest, um die Anforderungen für die Zeitsynchronisierung zu erfüllen.

Nehmen Sie die Rollenvariablenwerte in eine Datei mit dem Namen `timesync.yml` auf. Legen Sie die Datei `timesync.yml` im Unterverzeichnis `group_vars/all` ab, da diese Variablenwerte für alle Hosts im Inventar gelten.

- 4.1. Sehen Sie sich den Abschnitt *Role Variables* für die Rolle `rhel-system-roles.timesync` in der `README.md`-Datei an.

```
[student@workstation role-system]$ cat \
> /usr/share/doc/rhel-system-roles/timesync/README.md
...output omitted...
Role Variables
-----
...output omitted...
# List of NTP servers
timesync_ntp_servers:
  - hostname: foo.example.com      # Hostname or address of the server
    minpoll: 4                      # Minimum polling interval (default 6)
    maxpoll: 8                      # Maximum polling interval (default 10)
    iburst: yes                     # Flag enabling fast initial synchronization
                                    # (default no)
    pool: no                        # Flag indicating that each resolved address
                                    # of the hostname is a separate NTP server
                                    # (default no)
...output omitted...
# Name of the package which should be installed and configured for NTP.
# Possible values are "chrony" and "ntp". If not defined, the currently active
# or enabled service will be configured. If no service is active or enabled, a
# package specific to the system and its version will be selected.
timesync_ntp_provider: chrony
...output omitted...
```

- 4.2. Erstellen Sie das Unterverzeichnis `group_vars/all`.

```
[student@workstation role-system]$ mkdir -pv group_vars/all
mkdir: created directory 'group_vars'
mkdir: created directory 'group_vars/all'
```

- 4.3. Erstellen Sie die Datei `group_vars/all/timesync.yml` in einem Texteditor. Fügen Sie Variablendefinitionen entsprechend den Zeitsynchronisierungsanforderungen hinzu. Die Datei enthält jetzt Folgendes:

```
...
#rhel-system-roles.timesync variables for all hosts

timesync_ntp_provider: chrony

timesync_ntp_servers:
  - hostname: classroom.example.com
    iburst: yes
```

- 5. Fügen Sie `configure_time.yml` eine Aufgabe hinzu, um die Zeitzone für jeden Host festzulegen. Stellen Sie sicher, dass die Aufgabe das Modul `timezone` verwendet und nach der Rolle `rhel-system-roles.timesync` ausgeführt wird.

Verwenden Sie eine Variable (`host_timezone`) für den Namen der Zeitzone, da die Hosts nicht zur selben Zeitzone gehören.

- 5.1. Sehen Sie sich den Abschnitt *Examples* in der Dokumentation zum Modul `timezone` an.

```
[student@workstation role-system]$ ansible-doc timezone | grep -A 4 "EXAMPLES"
EXAMPLES:
```

```
- name: set timezone to Asia/Tokyo
  timezone:
    name: Asia/Tokyo
```

- 5.2. Fügen Sie dem Abschnitt `post_tasks` des Plays im Playbook `configure_time.yml` eine Aufgabe hinzu. Modellieren Sie die Aufgabe nach dem Beispiel in der Dokumentation, verwenden Sie jedoch die Variable `host_timezone` für den Namen der Zeitzone.

Die Dokumentation in `ansible-doc timezone` empfiehlt einen Neustart des Cron-Service, wenn das Modul die Zeitzone ändert, um sicherzustellen, dass Cron-Jobs zum richtigen Zeitpunkt ausgeführt werden. Da die Systemprotokollierung und andere Dienste die Systemzeitzone verwenden, müssen Sie jeden Host neu starten, wenn die Zeitzone geändert wird. Fügen Sie der Aufgabe das Keyword `notify` mit dem zugehörigen Wert `reboot host` hinzu. Der Abschnitt `post_tasks` des Plays sollte wie folgt aussehen:

```
post_tasks:
  - name: Set timezone
    timezone:
      name: "{{ host_timezone }}"
    notify: reboot host
```

- 5.3. Fügen Sie dem Play `Time Synchronization` den Handler `reboot host` hinzu. Das vollständige Playbook enthält nun Folgendes:

```
---
- name: Time Synchronization
  hosts: database_servers

  roles:
    - rhel-system-roles.timesync

  post_tasks:
    - name: Set timezone
      timezone:
        name: "{{ host_timezone }}"
      notify: reboot host
```

**Kapitel 7 |** Playbooks mit Rollen vereinfachen

```
handlers:
  - name: reboot host
    reboot:
```

- 6. Erstellen Sie für jedes Rechenzentrum eine Datei mit dem Namen `timezone.yml`, die einen geeigneten Wert für die Variable `host_timezone` enthält. Ermitteln Sie mit dem Befehl `timedatectl list-timezones` die gültige Zeitzonenzeichenfolge für jedes Rechenzentrum.

- 6.1. Erstellen Sie die `group_vars`-Unterverzeichnisse für die Hostgruppen `na-datacenter` und `europe-datacenter`.

```
[student@workstation role-system]$ mkdir -pv \
> group_vars/{na_datacenter,europe_datacenter}
mkdir: created directory 'group_vars/na_datacenter'
mkdir: created directory 'group_vars/europe_datacenter'
```

- 6.2. Ermitteln Sie mit dem Befehl `timedatectl list-timezones` die Zeitzone für das US-amerikanische und das europäische Rechenzentrum:

```
[student@workstation role-system]$ timedatectl list-timezones | grep Chicago
America/Chicago
[student@workstation role-system]$ timedatectl list-timezones | grep Helsinki
Europe/Helsinki
```

- 6.3. Erstellen Sie die Datei `timezone.yml` für beide Rechenzentren:

```
[student@workstation role-system]$ echo "host_timezone: America/Chicago" > \
> group_vars/na_datacenter/timezone.yml
[student@workstation role-system]$ echo "host_timezone: Europe/Helsinki" > \
> group_vars/europe_datacenter/timezone.yml
```

- 7. Führen Sie das Playbook aus.

```
[student@workstation role-system]$ ansible-playbook configure_time.yml

PLAY [Time Synchronization] ****
TASK [Gathering Facts] ****
ok: [serverb.lab.example.com]
ok: [servera.lab.example.com]

TASK [rhel-system-roles.timesync : Check if only NTP is needed] ****
ok: [servera.lab.example.com]
ok: [serverb.lab.example.com]

...output omitted...

TASK [rhel-system-roles.timesync : Enable timemaster] ****
skipping: [servera.lab.example.com]
skipping: [serverb.lab.example.com]
```

## Kapitel 7 | Playbooks mit Rollen vereinfachen

```
RUNNING HANDLER [rhel-system-roles.timesync : restart chronyd] ****
changed: [servera.lab.example.com]
changed: [serverb.lab.example.com]

TASK [Set timezone] ****
changed: [serverb.lab.example.com]
changed: [servera.lab.example.com]

RUNNING HANDLER [reboot host] ****
changed: [serverb.lab.example.com]
changed: [servera.lab.example.com]

servera.lab.example.com    : ok=17    changed=6      unreachable=0      failed=0
    skipped=20   rescued=0    ignored=6
serverb.lab.example.com    : ok=17    changed=6      unreachable=0      failed=0
    skipped=20   rescued=0    ignored=6
```

- 8. Verifizieren Sie die Zeitzoneneinstellungen jedes Servers. Verwenden Sie einen Ansible-Ad-hoc-Befehl, um die Ausgabe des Befehls `date` für alle Datenbankserver anzuzeigen.



### Anmerkung

Die tatsächlich aufgelisteten Zeitzonen hängen von der Jahreszeit und davon ab, ob die Sommerzeit gilt.

```
[student@workstation role-system]$ ansible database_servers -m shell -a date
servera.lab.example.com | CHANGED | rc=0 >>
Fri Jul 16 17:38:40 CDT 2021
serverb.lab.example.com | CHANGED | rc=0 >>
Sat Jul 17 01:38:40 EEST 2021
```

Die Zeitzoneneinstellung jedes Servers entspricht seinem geografischen Standort.

## Beenden

Führen Sie den Befehl `lab role-system finish` aus, um den verwalteten Host zu bereinigen.

```
[student@workstation ~]$ lab role-system finish
```

Hiermit ist die angeleitete Übung beendet.

# Erstellen von Rollen

## Ziele

Am Ende dieses Abschnitts sollten Sie zu Folgendem in der Lage sein: Rolle im Projektverzeichnis eines Playbooks erstellen und diese als Teil eines Plays im Playbook ausführen.

## Der Rollenerstellungsprozess

Die Erstellung von Rollen in Ansible erfordert keine besonderen Entwickler-Tools. Die Erstellung und die Verwendung einer Rolle ist ein Vorgang aus drei Schritten:

1. Struktur des Rollenverzeichnisses erstellen.
2. Rolleninhalt definieren.
3. Rolle in einem Playbook verwenden.

## Erstellen der Struktur des Rollenverzeichnisses

Standardmäßig sucht Ansible nach Rollen in einem Unterverzeichnis namens `roles` in dem Verzeichnis, das Ihr Ansible-Playbook enthält. So können Sie Rollen mit dem Playbook und anderen unterstützenden Dateien speichern.

Wenn Ansible die Rolle dort nicht finden kann, werden die von der Ansible-Konfigurationseinstellung `roles_path` angegebenen Verzeichnisse der Reihe nach durchsucht. Diese Variable enthält eine durch Doppelpunkte getrennte Verzeichnisliste für Suchanfragen. Der Standardwert dieser Variablen lautet:

```
~/.ansible/roles:/usr/share/ansible/roles:/etc/ansible/roles
```

Auf diese Weise können Sie Rollen auf Ihrem System installieren, die von mehreren Projekten gemeinsam genutzt werden. Beispielsweise könnten Ihre eigenen Rollen in Ihrem Benutzerverzeichnis im Unterverzeichnis `~/ ansible/roles` installiert sein, und auf dem System können für alle Benutzer Rollen im Verzeichnis `/usr/share/ansible/roles` installiert sein.

Jede Rolle hat ein eigenes Verzeichnis mit einer standardisierten Verzeichnisstruktur. Die folgende Verzeichnisstruktur enthält beispielsweise die Dateien, die die Rolle `motd` definieren.

```
[user@host ~]$ tree roles/
roles/
└── motd
    ├── defaults
    │   └── main.yml
    ├── files
    ├── handlers
    ├── meta
    │   └── main.yml
    ├── README.md
    └── tasks
```

```
|   └── main.yml
└── templates
    └── motd.j2
```

Die Datei `README.md` enthält eine grundlegende, für Benutzer lesbare Beschreibung der Rolle, die Dokumentation und Verwendungsbeispiele und Nicht-Ansible-Anforderungen, die erforderlich sein könnten. Das Unterverzeichnis `meta` enthält eine `main.yml`-Datei, die Informationen zu Autor, Lizenz, Kompatibilität und Abhängigkeiten für das Modul enthält. Das Unterverzeichnis `files` enthält inhaltskonstante Dateien, und das Unterverzeichnis `templates` enthält Vorlagen, die von jeder Rolle bei Bedarf eingesetzt werden können. Die anderen Unterverzeichnisse können `main.yml`-Dateien enthalten, die standardmäßige Variablenwerte, Handler, Aufgaben, Rollen-Metadaten oder Variablen definieren, die unabhängig vom jeweiligen Unterverzeichnis sind.

Wenn ein leeres Unterverzeichnis vorhanden ist, wie im Beispiel `handlers`, dann wird es ignoriert. Wenn eine Rolle keine Funktion verwendet, kann das Unterverzeichnis vollständig weggelassen werden. Zum Beispiel wurde das Unterverzeichnis `vars` in diesem Beispiel weggelassen.

## Erstellen eines Rollengerüsts

Sie können alle für eine neue Rolle erforderlichen Unterverzeichnisse und Dateien mit Linux-Standardbefehlen erstellen. Alternativ gibt es Befehlszeilen-Dienstprogramme, um den Prozess der Erstellung neuer Rollen zu automatisieren.

Mit dem Befehlszeilentool `ansible-galaxy` (wird später in diesem Kurs ausführlicher behandelt) werden Ansible-Rollen verwaltet, einschließlich der Erstellung neuer Rollen. Sie können `ansible-galaxy init` ausführen, um die Verzeichnisstruktur für eine neue Rolle zu erstellen. Geben Sie den Namen der Rolle als Argument für den Befehl an, wodurch ein Unterverzeichnis für die neue Rolle im aktuellen Arbeitsverzeichnis erstellt wird.

```
[user@host playbook-project]$ cd roles
[user@host roles]$ ansible-galaxy init my_new_role
- my_new_role was created successfully
[user@host roles]$ ls my_new_role/
defaults  files  handlers  meta  README.md  tasks  templates  tests  vars
```

## Definieren von Rolleninhalt

Nachdem Sie die Verzeichnisstruktur erstellt haben, müssen Sie den Inhalt der Rolle schreiben. Ein guter Ausgangspunkt dafür ist die Aufgabendatei `ROLENNAME/tasks/main.yml`, also die Hauptliste der von der Rolle ausgeführten Aufgaben.

Die Datei `tasks/main.yml` verwaltet die Datei `/etc/motd` auf verwalteten Hosts. Sie verwendet das Modul `template`, um die Vorlage `motd.j2` auf dem verwalteten Host bereitzustellen. Weil das Modul `template` in einer Rollenaufgabe konfiguriert wird anstatt in einer Playbook-Aufgabe, wird die Vorlage `motd.j2` aus dem Unterverzeichnis `templates` der Rolle abgerufen.

```
[user@host ~]$ cat roles/motd/tasks/main.yml
---
# tasks file for motd

- name: deliver motd file
  template:
    src: motd.j2
```

## Kapitel 7 | Playbooks mit Rollen vereinfachen

```
dest: /etc/motd
owner: root
group: root
mode: 0444
```

Der folgende Befehl zeigt den Inhalt der Vorlage `motd.j2` der Rolle `motd` an. Er bezieht sich auf Ansible-Fakten und auf die Variable `system_owner`.

```
[user@host ~]$ cat roles/motd/templates/motd.j2
This is the system {{ ansible_facts['hostname'] }}.

Today's date is: {{ ansible_facts['date_time']['date'] }}.

Only use this system with permission.
You can ask {{ system_owner }} for access.
```

Die Rolle definiert einen Standardwert für die Variable `system_owner`. Dieser Wert ist in der Datei `defaults/main.yml` in der Verzeichnisstruktur der Rolle festgelegt.

Die folgende Datei `defaults/main.yml` legt die Variable `system_owner` auf `user@host.example.com` fest. Dies ist die E-Mail-Adresse, die in der Datei `/etc/motd` von verwalteten Hosts hinterlegt ist, für die diese Rolle angewendet wird.

```
[user@host ~]$ cat roles/motd/defaults/main.yml
---
system_owner: user@host.example.com
```

## Empfohlene Vorgehensweisen für die Entwicklung von Rolleninhalten

Rollen ermöglichen das modulare Schreiben von Playbooks. Um die Wirksamkeit neu entwickelter Rollen zu erhöhen, sollten Sie die folgenden empfohlenen Vorgehensweisen bei Ihrer Rollenentwicklung umsetzen:

- Verwalten Sie jede Rolle in einem eigenen Repository für die Versionskontrolle. Ansible funktioniert gut mit Git-basierten Repositorys.
- Vertrauliche Informationen wie Passwörter oder SSH-Schlüssel sollten nicht im Rollen-Repository gespeichert werden. Vertrauliche Werte sollten als Variablen mit nicht vertraulichen Standardwerten parametrisiert werden. Playbooks, die die Rolle verwenden, sind für die Definition vertraulicher Variablen durch Ansible Vault-Variablen dateien, Umgebungsvariablen oder andere `ansible-playbook`-Optionen verantwortlich.
- Beginnen Sie Ihre Rolle mit `ansible-galaxy init`, und entfernen Sie alle Verzeichnisse und Dateien, die Sie nicht benötigen.
- Erstellen und verwalten Sie `README.md`- und `meta/main.yml`-Dateien, um zu dokumentieren, was Ihre Rolle ausführt, wer sie geschrieben hat und wie sie verwendet wird.
- Konzentrieren Sie Ihre Rolle auf einen bestimmten Zweck oder eine bestimmte Funktion. Anstatt eine Rolle für viele Aufgaben zu nutzen, könnten Sie mehrere Rollen schreiben.
- Verwenden Sie Rollen häufig wieder und gestalten Sie sie um. Erstellen Sie keine neuen Rollen für Randkonfigurationen. Wenn eine vorhandene Rolle den Großteil

## Kapitel 7 | Playbooks mit Rollen vereinfachen

der erforderlichen Konfiguration erfüllt, gestalten Sie die vorhandene Rolle um, um das neue Konfigurationsszenario zu integrieren. Verwenden Sie Integrations- und Regressionstesttechniken, um sicherzustellen, dass die Rolle die erforderliche neue Funktionalität bietet und auch für vorhandene Playbooks keine Probleme verursacht.

## Definieren von Rollenabhängigkeiten

Rollenabhängigkeiten ermöglichen einer Rolle andere Rollen als Abhängigkeiten einzubeziehen. Beispielsweise kann eine Rolle, die einen Dokumentationsserver definiert, von einer anderen Rolle, die einen Webserver installiert und konfiguriert, abhängig sein. Abhängigkeiten sind in der Datei `meta/main.yml` in der Rollenverzeichnishierarchie definiert.

Die folgende Datei `meta/main.yml` dient als Beispiel.

```
---
dependencies:
  - role: apache
    port: 8080
  - role: postgres
    dbname: serverlist
    admin_user: felix
```

Standardmäßig werden Rollen als Abhängigkeit nur einmal einem Playbook hinzugefügt. Wenn eine andere Rolle diese als Abhängigkeit aufführt, dann wird diese nicht noch einmal ausgeführt. Dieses Verhalten kann durch das Festlegen der Variable `allow_duplicates` auf `yes` in der Datei `meta/main.yml` überschrieben werden.



### Wichtig

Begrenzen Sie die Abhängigkeiten Ihrer Rolle von anderen Rollen. Abhängigkeiten erschweren die Pflege Ihrer Rolle, insbesondere wenn es sich um viele komplexe Abhängigkeiten handelt.

## Verwenden der Rolle in einem Playbook

Um auf eine Rolle zuzugreifen, verweisen Sie auf diese im Abschnitt `roles:` eines Plays. Das folgende Playbook verweist auf die Rolle `motd`. Weil keine Variablen spezifiziert sind, wird die Rolle mit standardmäßigen Variablenwerten angewendet.

```
[user@host ~]$ cat use-motd-role.yml
---
- name: use motd role playbook
  hosts: remote.example.com
  remote_user: devops
  become: true
  roles:
    - motd
```

Wenn das Playbook ausgeführt wird, werden aufgrund der Rolle ausgeführte Aufgaben durch das Präfix für Rollennamen identifiziert. Die folgende Beispieldaten veranschaulicht dies mit dem Präfix `motd :` im Aufgabennamen:

```
[user@host ~]$ ansible-playbook -i inventory use-motd-role.yml

PLAY [use motd role playbook] ****

TASK [setup] ****
ok: [remote.example.com]

TASK [motd: deliver motd file] ****
changed: [remote.example.com]

PLAY RECAP ****
remote.example.com : ok=2    changed=1    unreachable=0    failed=0
```

Das obige Szenario geht davon aus, dass sich die Rolle `motd` im Verzeichnis `roles` befindet. Im weiteren Verlauf des Kurses erfahren Sie, wie Sie eine Rolle verwenden, die sich `remote` in einem Repository für die Versionskontrolle befindet.

## Ändern des Rollenverhaltens mit Variablen

Eine gut geschriebene Rolle verwendet Standardvariablen, um das Verhalten der Rolle an ein zugehöriges Konfigurationsszenario anzupassen. Dies trägt dazu bei, die Rolle in verschiedenen Kontexten allgemeiner und wiederverwendbarer zu gestalten.

Der Wert einer Variablen, die im Verzeichnis `defaults` in einer Rolle definiert ist, wird überschrieben, wenn dieselbe Variable folgendermaßen definiert ist:

- in einer Inventardatei, entweder als Hostvariable oder als Gruppenvariable
- in einer YAML-Datei in den Verzeichnissen `group_vars` oder `host_vars` eines Playbook-Projekts
- als im Keyword `vars` verschachtelte Variable eines Plays
- als Variable, wenn die Rolle in das Keyword `roles` eines Plays einbezogen wird

Das folgende Beispiel zeigt die Verwendung der Rolle `motd` mit einem anderen Wert für die Rollenvariable `system_owner`. Der festgelegte Wert `someone@host.example.com` ersetzt die Variablenreferenz, wenn die Rolle für einen verwalteten Host verwendet wird.

```
[user@host ~]$ cat use-motd-role.yml
---
- name: use motd role playbook
  hosts: remote.example.com
  remote_user: devops
  become: true
  vars:
    system_owner: someone@host.example.com
  roles:
    - role: motd
```

Bei dieser Definition ersetzt die Variable `system_owner` den Wert der Standardvariablen mit demselben Namen. Alle Variablendefinitionen, die im Keyword `vars` verschachtelt sind, ersetzen den Wert derselben Variablen nicht, wenn sie im Verzeichnis `vars` der Rolle definiert sind.

## Kapitel 7 | Playbooks mit Rollen vereinfachen

Das folgende Beispiel zeigt auch die Verwendung der Rolle `motd` mit einem anderen Wert für die Rollenvariable `system_owner`. Der angegebene Wert `someone@example.com` ersetzt die Variablenreferenz, unabhängig davon, ob sie im Verzeichnis `vars` oder `defaults` der Rolle definiert ist.

```
[user@host ~]$ cat use-motd-role.yml
---
- name: use motd role playbook
  hosts: remote.example.com
  remote_user: devops
  become: true
  roles:
    - role: motd
      system_owner: someone@example.com
```



### Wichtig

Variablenpriorität kann verwirrend sein, wenn Sie mit Rollenvariablen in einem Play arbeiten.

- Fast jede andere Variable überschreibt die Standardvariablen einer Rolle: Inventarvariablen, Play-`vars`, Inline-Rollenparameter und so weiter.
- Ein kleinerer Teil der Variablen kann Variablen überschreiben, die im Verzeichnis `vars` einer Rolle definiert sind. Fakten, mit `include_vars` geladene Variablen, registrierte Variablen und Rollenparameter gehören zu diesen Variablen. Inventarvariablen und Play-`vars` gehören nicht dazu. Das ist wichtig, da es dazu beiträgt, dass Ihr Play nicht versehentlich die internen Funktionen der Rolle beeinträchtigt.
- Variablen, die inline als Rollenparameter deklariert sind, wie im letzten der vorherigen Beispiele, haben allerdings eine sehr hohe Priorität. Sie können Variablen überschreiben, die im Verzeichnis `vars` einer Rolle definiert sind. Wenn ein Rollenparameter den gleichen Namen wie eine Variable hat, die in Play-`vars` oder in der `vars` einer Rolle festgelegt ist oder einer Inventar- oder Playbook-Variablen entspricht, überschreibt der Rollenparameter die andere Variable.



### Literaturhinweise

#### Using Roles – Ansible-Dokumentation

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_reuse\\_roles.html#using-roles](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_reuse_roles.html#using-roles)

#### Using Variables – Ansible-Dokumentation

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_variables.html](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_variables.html)

## ► Angeleitete Übung

# Erstellen von Rollen

In dieser Übung erstellen Sie Ansible-Rollen, die Variablen, Dateien, Vorlagen, Aufgaben und Handler nutzen, um einen Netzwerkservice bereitzustellen.

## Ergebnisse

Sie sollten in der Lage sein, eine Rolle zu erstellen, die Variablen und Parameter verwenden.

Die Rolle `myvhost` installiert und konfiguriert den Apache-Service auf einem Host. Eine Vorlage mit dem Namen `vhost.conf.j2` wird bereitgestellt, die zum Erstellen von `/etc/httpd/conf.d/vhost.conf` verwendet wird.

## Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab role-create start` aus. Damit wird das Arbeitsverzeichnis `/home/student/role-create` erstellt und darin eine Ansible-Konfigurationsdatei und das Hostinventar gespeichert.

```
[student@workstation ~]$ lab role-create start
```

## Anweisungen

- 1. Wechseln Sie in das Arbeitsverzeichnis `/home/student/role-create`.

```
[student@workstation ~]$ cd ~/role-create  
[student@workstation role-create]$
```

- 2. Erstellen Sie die Verzeichnisstruktur für eine Rolle namens `myvhost`. Diese Rolle beinhaltet festgelegte Dateien, Vorlagen, Aufgaben und Handler.

```
[student@workstation role-create]$ mkdir -v roles; cd roles  
mkdir: created directory 'roles'  
[student@workstation roles]$ ansible-galaxy init myvhost  
- myvhost was created successfully  
[student@workstation roles]$ rm -rfv myvhost/{defaults,vars,tests}  
removed 'myvhost/defaults/main.yml'  
removed directory: 'myvhost/defaults'  
removed 'myvhost/vars/main.yml'  
removed directory: 'myvhost/vars'  
removed 'myvhost/tests/inventory'  
removed 'myvhost/tests/test.yml'  
removed directory: 'myvhost/tests'  
[student@workstation roles]$ cd ..  
[student@workstation role-create]$
```

- 3. Bearbeiten Sie die Datei `main.yml` im Unterverzeichnis `tasks` der Rolle. Die Rolle sollte die folgenden Aufgaben durchführen:
- Das Paket `httpd` ist installiert.
  - Der Service `httpd` ist gestartet und aktiviert.
  - Die Webserver-Konfigurationsdatei wird mit einer von der Rolle bereitgestellten Vorlage installiert.
- 3.1. Bearbeiten Sie die Datei `roles/myvhost/tasks/main.yml`. Integrieren Sie den Code zur Nutzung von Modul `yum`, um das Paket `httpd` zu installieren. Die Datei sollte folgenden Inhalt haben:

```
---
```

```
# tasks file for myvhost
```

```
- name: Ensure httpd is installed
```

```
  yum:
```

```
    name: httpd
```

```
    state: latest
```

- 3.2. Fügen Sie der Datei `roles/myvhost/tasks/main.yml` zusätzlichen Code hinzu, um mit dem Modul `service` den Service `httpd` zu starten und zu aktivieren.

```
- name: Ensure httpd is started and enabled
```

```
  service:
```

```
    name: httpd
```

```
    state: started
```

```
    enabled: true
```

- 3.3. Fügen Sie einen weiteren Absatz hinzu, um das Modul `template` zur Erstellung von `/etc/httpd/conf.d/vhost.conf` auf dem verwalteten Host zu verwenden. Ein Handler wird erforderlich sein, um den Daemon `httpd` neu zu starten, wenn diese Datei aktualisiert ist.

```
- name: vhost file is installed
```

```
  template:
```

```
    src: vhost.conf.j2
```

```
    dest: /etc/httpd/conf.d/vhost.conf
```

```
    owner: root
```

```
    group: root
```

```
    mode: 0644
```

```
  notify:
```

```
    - restart httpd
```

- 3.4. Speichern Sie Ihre Änderungen und schließen Sie die Datei `roles/myvhost/tasks/main.yml`.
- 4. Erstellen Sie einen Handler zum Neustart des Service `httpd`. Bearbeiten Sie die Datei `roles/myvhost/handlers/main.yml`, und fügen Sie Code für die Verwendung des Moduls `service` hinzu. Speichern und schließen Sie die Datei anschließend. Die Datei sollte folgenden Inhalt haben:

```
---  
# handlers file for myvhost  
  
- name: restart httpd  
  service:  
    name: httpd  
    state: restarted
```

- 5. Verschieben Sie die Vorlage `vhost.conf.j2` aus dem Projektverzeichnis in das Unterverzeichnis `templates` der Rolle.

```
[student@workstation role-create]$ mv -v vhost.conf.j2 roles/myvhost/templates/  
renamed 'vhost.conf.j2' -> 'roles/myvhost/templates/vhost.conf.j2'
```

- 6. Erstellen Sie den HTML-Inhalt, der vom Webserver bereitgestellt werden soll.

- 6.1. Erstellen Sie das Verzeichnis `files/html/`, in dem der Inhalt gespeichert wird.

```
[student@workstation role-create]$ mkdir -pv files/html  
mkdir: created directory 'files/html'
```

- 6.2. Erstellen Sie eine Datei `index.html` unter dem Verzeichnis mit den Inhalten:  
`simple index`.

```
[student@workstation role-create]$ echo \  
> 'simple index' > files/html/index.html
```

- 7. Prüfen Sie die Rolle `myvhost`, um eine korrekte Funktionsweise zu gewährleisten.

- 7.1. Erstellen Sie ein Playbook mit dem Namen `use-vhost-role.yml`, das diese Rolle verwendet. Fügen Sie eine Aufgabe hinzu, um den HTML-Inhalt aus `files/html/` zu kopieren. Verwenden Sie das Modul `copy` und fügen Sie einen Schrägstrich nach dem Namen des Quellverzeichnisses hinzu. Das Playbook sollte Folgendes enthalten:

```
---  
- name: Use myvhost role playbook  
  hosts: webservers  
  pre_tasks:  
    - name: pre_tasks message  
      debug:  
        msg: 'Ensure web server configuration.'  
  
  roles:  
    - myvhost  
  
  post_tasks:  
    - name: HTML content is installed  
      copy:  
        src: files/html/  
        dest: "/var/www/vhosts/{{ ansible_hostname }}"
```

```
- name: post_tasks message
  debug:
    msg: 'Web server is configured.'
```



### Anmerkung

Der angehängte Schrägstrich bewirkt, dass das Quellverzeichnis und sein gesamter Inhalt auf den verwalteten Host kopiert werden.

- 7.2. Verifizieren Sie vor Ausführung des Playbooks, dass dessen Syntax korrekt ist, indem Sie `ansible-playbook` mit der Option `--syntax-check` ausführen. Wenn Fehler gemeldet werden, korrigieren Sie sie, bevor Sie mit dem nächsten Schritt fortfahren. Es sollte ungefähr folgende Ausgabe angezeigt werden:

```
[student@workstation role-create]$ ansible-playbook use-vhost-role.yml \
> --syntax-check

playbook: use-vhost-role.yml
```

- 7.3. Führen Sie das Playbook aus. Prüfen Sie die Ausgabe, um zu bestätigen, dass Ansible die Aktionen auf dem Webserver `servera` ausgeführt hat.

```
[student@workstation role-create]$ ansible-playbook use-vhost-role.yml

PLAY [Use myvhost role playbook] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [pre_tasks message] ****
ok: [servera.lab.example.com] => {
    "msg": "Ensure web server configuration."
}

TASK [myvhost : Ensure httpd is installed] ****
changed: [servera.lab.example.com]

TASK [myvhost : Ensure httpd is started and enabled] ****
changed: [servera.lab.example.com]

TASK [myvhost : vhost file is installed] ****
changed: [servera.lab.example.com]

RUNNING HANDLER [myvhost : restart httpd] ****
changed: [servera.lab.example.com]

TASK [HTML content is installed] ****
changed: [servera.lab.example.com]

TASK [post_tasks message] ****
ok: [servera.lab.example.com] => {
    "msg": "Web server is configured."
}
```

```
PLAY RECAP ****
servera.lab.example.com : ok=8    changed=5    unreachable=0    failed=0
```

- 7.4. Führen Sie die Ad-hoc-Befehle aus, um zu bestätigen, dass die Rolle funktioniert. Das Paket `httpd` sollte installiert sein und der Service `httpd` ausgeführt werden.

```
[student@workstation role-create]$ ansible webservers -a \
> 'systemctl is-active httpd'
servera.lab.example.com | CHANGED | rc=0 >>
active

[student@workstation role-create]$ ansible webservers -a \
> 'systemctl is-enabled httpd'
servera.lab.example.com | CHANGED | rc=0 >>
enabled
```

- 7.5. Die Apache-Konfiguration sollte mit den erweiterten Vorlagevariablen installiert sein.

```
[student@workstation role-create]$ ansible webservers -a \
> 'cat /etc/httpd/conf.d/vhost.conf'
servera.lab.example.com | CHANGED | rc=0 >>
# Ansible managed:

<VirtualHost *:80>
    ServerAdmin webmaster@servera.lab.example.com
    ServerName servera.lab.example.com
    ErrorLog logs/servera-error.log
    CustomLog logs/servera-common.log common
    DocumentRoot /var/www/vhosts/servera/

    <Directory /var/www/vhosts/servera/>
        Options +Indexes +FollowSymlinks +Includes
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>
```

- 7.6. Der HTML-Inhalt befindet sich im Verzeichnis mit dem Namen `/var/www/vhosts/servera`. Die Datei `index.html` sollte die Zeichenfolge „simple index“ enthalten.

```
[student@workstation role-create]$ ansible webservers -a \
> 'cat /var/www/vhosts/servera/index.html'
servera.lab.example.com | CHANGED | rc=0 >>
simple index
```

- 7.7. Verwenden Sie das Modul `uri` in einem Ad-hoc-Befehl, um zu prüfen, ob der Webinhalt lokal verfügbar ist. Legen Sie den Parameter `return_content` auf `true` fest, damit der Inhalt der Antwort des Servers zur Ausgabe hinzugefügt wird. Der Serverinhalt sollte die Zeichenfolge `simple index\n` sein.

```
[student@workstation role-create]$ ansible webservers -m uri \
> -a 'url=http://localhost return_content=true'
servera.lab.example.com | SUCCESS => {
    "accept_ranges": "bytes",
    "changed": false,
    "connection": "close",
    "content": "simple index\n",
    ...output omitted...
    "status": 200,
    "url": "http://localhost"
}
```

7.8. Überprüfen Sie, ob der Inhalt des Webservers für Remote-Clients verfügbar ist.

```
[student@workstation role-create]$ curl http://servera.lab.example.com
simple index
```

## Beenden

Führen Sie den Befehl `lab role-create finish` aus, um den verwalteten Host zu bereinigen.

```
[student@workstation ~]$ lab role-create finish
```

Hiermit ist die angeleitete Übung beendet.

# Bereitstellen von Rollen mit Ansible Galaxy

---

## Ziele

Am Ende dieses Abschnitts sollten Sie in der Lage sein, Rollen aus Ansible Galaxy oder anderen Quellen, beispielsweise aus einem Git-Repository, auszuwählen und abzurufen und sie in Playbooks zu verwenden.

## Einführung in Ansible Galaxy

Ansible Galaxy [<https://galaxy.ansible.com>] ist eine öffentliche Library mit Ansible-Inhalt, der von einer Vielzahl Administratoren und Benutzern von Ansible geschrieben wurde. Sie enthält tausende Ansible-Rollen und eine Datenbank mit Suchfunktion, sodass Ansible-Benutzer Rollen identifizieren und administrative Aufgaben durchführen können. Ansible Galaxy enthält Links zu Dokumentation und Videos für neue Ansible-Benutzer und Rollenentwickler.

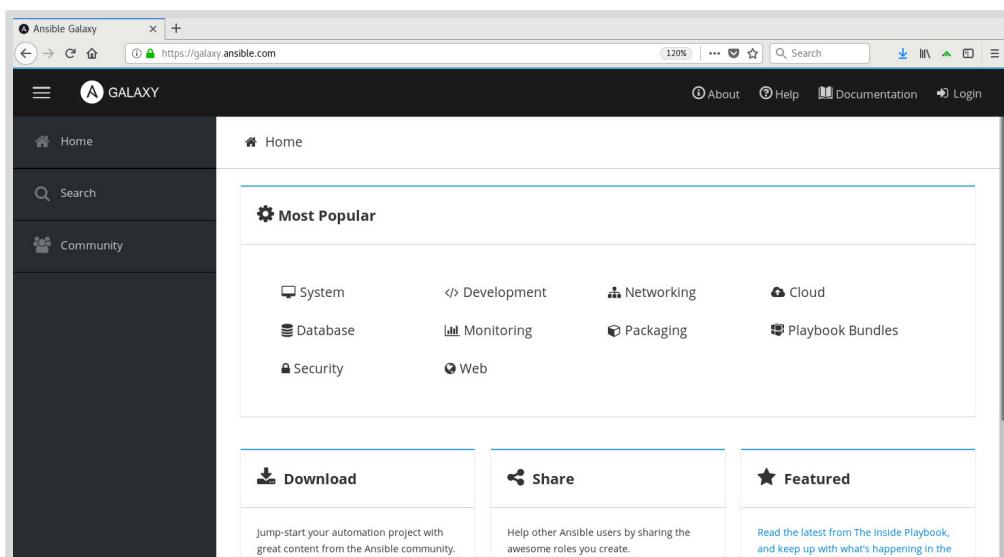


Abbildung 7.1: Homepage von Ansible Galaxy

Sie können mit dem Befehl `ansible-galaxy`, mit dem Sie Rollen von Ansible Galaxy abrufen und verwalten, auch Rollen, die Ihre Projekte benötigen, aus eigenen Git-Repositorys abrufen und verwalten.

## Hilfe zu Ansible Galaxy

Die Registerkarte **Documentation** auf der Homepage der Ansible Galaxy-Website führt zu einer Seite, auf der die Verwendung von Ansible Galaxy beschrieben ist. Es gibt Inhalte, die den Download und die Verwendung von Rollen in Ansible Galaxy beschreiben. Anweisungen zur Entwicklung von Rollen und zum Hochladen auf Ansible Galaxy finden Sie ebenso auf dieser Seite.

## Durchsuchen von Ansible Galaxy nach Rollen

Über die Registerkarte **Search** links auf der Homepage der Ansible Galaxy-Website können Benutzer auf Informationen über die auf Ansible Galaxy veröffentlichten Rollen zugreifen. Sie

## Kapitel 7 | Playbooks mit Rollen vereinfachen

können eine Ansible-Rolle nach Name, über Tags oder Rollenattribute suchen. Die Ergebnisse werden in absteigender Reihenfolge nach **Best Match** angezeigt: Dies ist ein berechneter Wert, der auf Rollenqualität, Rollenpopularität und Suchkriterien basiert.



### Anmerkung

„Content Scoring“ [[https://galaxy.ansible.com/docs/contributing/content\\_scoring.html](https://galaxy.ansible.com/docs/contributing/content_scoring.html)] in der Dokumentation bietet weitere Informationen dazu, wie Rollen von Ansible Galaxy bewertet werden.

The screenshot shows the Ansible Galaxy search interface. The left sidebar includes links for Home, Search (which is selected), Community, My Content, and My Imports. The main search area has a search bar with the placeholder 'Search for...' and a dropdown menu for 'Filters'. Below the search bar, it says '(18502 results)'. There are two search results listed:

- java role**: Java for Linux by geerlingguy. It has a 'build passing' status, a 5/5 score, 1280657 downloads, 9 watchers, 133 stars, and 173 forks. It was last imported 3 days ago with a Best Match of 0.5147. Associated tags include development, java, jdk, openjdk, oracle, system, web.
- docker role**: Docker for Linux by geerlingguy. It has a 'build passing' status, a 5/5 score, 1069555 downloads, 20 watchers, and 8 forks. It was last imported 3 days ago with a Best Match of 0.5147. Associated tags include compose, containers, docker.

To the right of the search results is a 'Popular Tags' sidebar with a list of tags and their counts:

Tag	Count
system	5,271
development	2,627
web	2,227
monitoring	1,172
networking	962
database	914
cloud	848
packaging	719
docker	608

Abbildung 7.2: Suchbildschirm von Ansible Galaxy

Ansible Galaxy zeigt an, wie oft die jeweilige Rolle von Ansible Galaxy heruntergeladen wurde. Darüber hinaus zeigt Ansible Galaxy auch die Anzahl der Beobachter, Gabelungen und Sterne an, die das GitHub-Repository der Rolle hat. Benutzer können anhand dieser Informationen feststellen, wie aktiv die Entwicklung für eine Rolle ist und wie beliebt sie in der Community ist.

Die folgende Abbildung zeigt die Suchergebnisse, die Ansible Galaxy nach einer Keyword-Suche für `redis` anzeigt. Beachten Sie, dass das erste Ergebnis einen Wert für **Best Match** von 0.9009 hat.

## Kapitel 7 | Playbooks mit Rollen vereinfachen

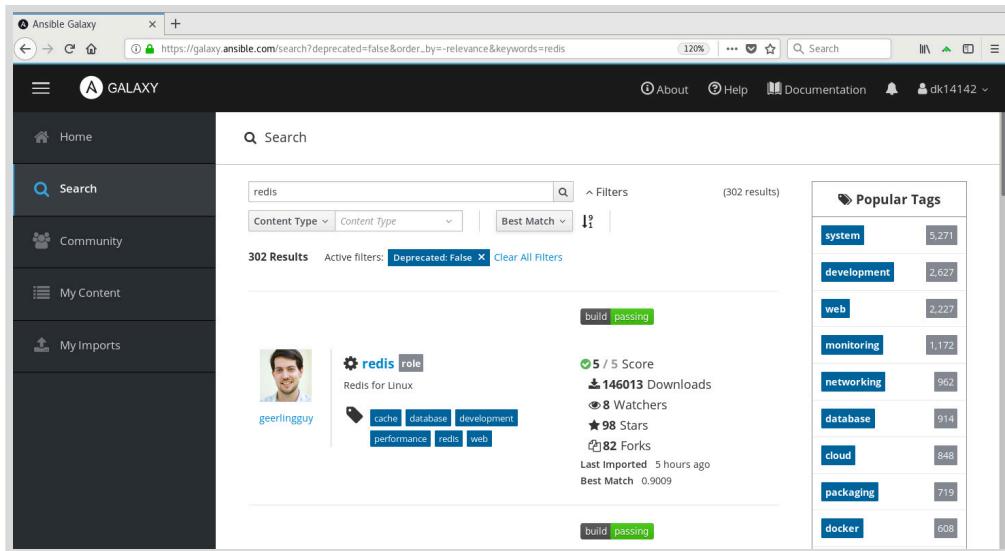


Abbildung 7.3: Beispiel für Ansible Galaxy-Suchergebnisse

Über das Pulldown-Menü **Filters** rechts neben dem Suchfeld können Sie Suchen für Keywords, Autoren-IDs, Plattform und Tags durchführen. Mögliche Plattformwerte umfassen unter anderem EL für Red Hat Enterprise Linux (und eng verwandte Distributionen wie CentOS) und Fedora.

Tags sind beliebige, vom Rollenautor festgelegte einzelne Wörter, die die Rolle beschreiben und kategorisieren. Benutzer können mithilfe von Tags relevante Rollen suchen. Zu möglichen Tag-Werten zählen: `system`, `development`, `web`, `packaging` und andere. Eine Rolle kann in Ansible Galaxy mit bis zu 20 Tags versehen sein.



### Wichtig

Auf der Ansible Galaxy-Suchoberfläche stimmen die Keyword-Suchvorgänge mit Wörtern oder Formulierungen in Datei `README`, im Inhaltsnamen oder in der Inhaltsbeschreibung überein. Im Gegensatz dazu stimmen Tag-Suchen speziell mit den vom Autor für die Rolle festgelegten Tag-Werten überein.

## Das Befehlszeilen-Tool von Ansible Galaxy

Das Befehlszeilen-Tool `ansible-galaxy` kann zum Suchen, Anzeigen weiterer Informationen, Installieren, Auflisten, Löschen oder Initialisieren von Rollen verwendet werden.

### Suchen von Rollen über die Befehlszeile

Der Unterbefehl `ansible-galaxy search` durchsucht Ansible Galaxy nach Rollen. Wenn Sie eine Zeichenfolge als Argument angeben, wird Ansible Galaxy nach Rollen per Keyword durchsucht. Sie können mit den Optionen `--author`, `--platforms` und `--galaxy-tags` die Suchergebnisse eingrenzen. Sie können diese Optionen auch als Hauptsuchschlüssel verwenden. Zum Beispiel zeigt der Befehl `ansible-galaxy search --author geerlingguy` alle vom Benutzer `geerlingguy` eingereichten Rollen an.

Die Ergebnisse werden in alphabetischer Reihenfolge, nicht absteigend nach `Best Match` angezeigt. Das folgende Beispiel zeigt die Namen der Rollen an, die `redis` enthalten und die für die Plattform Enterprise Linux (EL) verfügbar sind.

```
[user@host ~]$ ansible-galaxy search 'redis' --platforms EL

Found 124 roles matching your search:

  Name          Description
  ----
  lit.sudo      Ansible role for managing sudoers
  AerisCloud.librato  Install and configure the Librato Agent
  AerisCloud.redis  Installs redis on a server
  AlbanAndrieu.java  Manage Java installation
  andrewrothstein.redis  builds Redis from src and installs
  ...output omitted...
  geerlingguy.php-redis  PhpRedis support for Linux
  geerlingguy.redis  Redis for Linux
  gikoluo.filebeat  Filebeat for Linux.
  ...output omitted...
```

Der Unterbefehl `ansible-galaxy info` zeigt weitere Informationen zur Rolle an. Ansible Galaxy ruft diese Informationen von verschiedenen Stellen ab, einschließlich von der Datei `meta/main.yml` der Rolle und ihrem GitHub-Repository. Der folgende Befehl zeigt Informationen über die Rolle `geerlingguy.redis` an, die auf Ansible Galaxy verfügbar ist.

```
[user@host ~]$ ansible-galaxy info geerlingguy.redis

Role: geerlingguy.redis
      description: Redis for Linux
      active: True
  ...output omitted...
      download_count: 146209
      forks_count: 82
      github_branch: master
      github_repo: ansible-role-redis
      github_user: geerlingguy
  ...output omitted...
      license: license (BSD, MIT)
      min_ansible_version: 2.4
      modified: 2018-11-19T14:53:29.722718Z
      open_issues_count: 11
      path: [u'/etc/ansible/roles', u'/usr/share/ansible/roles']
      role_type: ANS
      stargazers_count: 98
  ...output omitted...
```

## Installieren von Rollen von Ansible Galaxy

Der Unterbefehl `ansible-galaxy install` lädt eine Rolle von Ansible Galaxy herunter und installiert diese dann lokal auf dem Kontrollknoten.

Standardmäßig werden Rollen im ersten Verzeichnis im `roles_path` des Benutzers installiert, das beschreibbar ist. Basierend auf der für Ansible festgelegten Standardeinstellung `roles_path` wird die Rolle normalerweise im Verzeichnis `~/.ansible/roles` des Benutzers installiert. Der Standard-`roles_path` wird möglicherweise von Ihrer aktuellen Ansible-Konfigurationsdatei

## Kapitel 7 | Playbooks mit Rollen vereinfachen

oder von der Umgebungsvariablen `ANSIBLE_ROLES_PATH` überschrieben, die das Verhalten von `ansible-galaxy` beeinflusst.

Sie können auch mit der Option `-p DIRECTORY` ein bestimmtes Verzeichnis angeben, in dem die Rolle installiert werden soll.

Im folgenden Beispiel installiert `ansible-galaxy` die Rolle `geerlingguy.redis` im Verzeichnis `roles` eines Playbook-Projekts. Das aktuelle Arbeitsverzeichnis des Befehls ist `/opt/project`.

```
[user@host project]$ ansible-galaxy install geerlingguy.redis -p roles/
- downloading role 'redis', owned by geerlingguy
- downloading role from https://github.com/geerlingguy/...output omitted...
- extracting geerlingguy.redis to /opt/project/roles/geerlingguy.redis
- geerlingguy.redis (1.6.0) was installed successfully
[user@host project]$ ls roles/
geerlingguy.redis
```

## Installieren von Rollen mit einer Anforderungsdatei

Sie können auch `ansible-galaxy` verwenden, um eine Liste von Rollen basierend auf Definitionen in einer Textdatei zu installieren. Wenn Sie beispielsweise über ein Playbook verfügen, für das bestimmte Rollen installiert sein müssen, können Sie eine `roles/requirements.yml`-Datei im Projektverzeichnis erstellen, die angibt, welche Rollen benötigt werden. Diese Datei dient als Abhängigkeitsmanifest für das Playbook-Projekt, mit dem Playbooks unabhängig von den unterstützenden Rollen entwickelt und getestet werden können.

Zum Beispiel könnte eine einfache `requirements.yml` zur Installation von `geerlingguy.redis` folgendermaßen lauten:

```
- src: geerlingguy.redis
  version: "1.5.0"
```

Das Attribut `src` gibt die Quelle der Rolle an, in diesem Fall die Rolle `geerlingguy.redis` von Ansible Galaxy. Das Attribut `version` ist optional und gibt die Version der zu installierenden Rolle, in diesem Fall `1.5.0`, an.



### Wichtig

Sie sollten insbesondere für Playbooks in der Produktion die Version der Rolle in Ihrer `requirements.yml`-Datei angeben.

Wenn Sie keine Version angeben, erhalten Sie die neueste Version der Rolle. Wenn der Upstream-Autor Änderungen an der Rolle vornimmt, die mit Ihrem Playbook nicht kompatibel sind, kann dies zu einem Automatisierungsfehler oder anderen Problemen führen.

Verwenden Sie die Option `-r REQUIREMENTS-FILE`, um die Rollen mithilfe einer Rollendatei zu installieren:

```
[user@host project]$ ansible-galaxy install -r roles/requirements.yml \
> -p roles
- downloading role 'redis', owned by geerlingguy
- downloading role from https://github.com/geerlingguy/ansible-role-redis/
archive/1.6.0.tar.gz
- extracting geerlingguy.redis to /opt/project/roles/geerlingguy.redis
- geerlingguy.redis (1.6.0) was installed successfully
```

Mit `ansible-galaxy` können Sie Rollen installieren, die sich nicht auf Ansible Galaxy befinden. Eigene proprietäre oder interne Rollen können Sie in einem privaten Git-Repository oder auf einem Webserver hosten. Das folgende Beispiel zeigt, wie Sie eine Anforderungsdatei mit einer Vielzahl von Remote-Quellen konfigurieren.

```
[user@host project]$ cat roles/requirements.yml
# from Ansible Galaxy, using the latest version
- src: geerlingguy.redis

# from Ansible Galaxy, overriding the name and using a specific version
- src: geerlingguy.redis
  version: "1.5.0"
  name: redis_prod

# from any Git-based repository, using HTTPS
- src: https://gitlab.com/guardianproject-ops/ansible-nginx-acme.git
  scm: git
  version: 56e00a54
  name: nginx-acme

# from any Git-based repository, using SSH
- src: git@gitlab.com:guardianproject-ops/ansible-nginx-acme.git
  scm: git
  version: master
  name: nginx-acme-ssh

# from a role tar ball, given a URL;
#   supports 'http', 'https', or 'file' protocols
- src: file:///opt/local/roles/myrole.tar
  name: myrole
```

Das Keyword `src` gibt den Namen der Ansible Galaxy-Rolle an. Wenn die Rolle nicht auf Ansible Galaxy gehostet wird, gibt das Keyword `src` die URL der Rolle an.

Wenn die Rolle in einem Quellcodeverwaltungs-Repository gehostet wird, ist das Attribut `scm` erforderlich. Der Befehl `ansible-galaxy` kann Rollen von Git-basierten oder Mercurial-basierten Software-Repositorien herunterladen und installieren. Für ein Git-basiertes Repository muss der `scm`-Wert `git` angegeben werden und für eine in einem Mercurial-Repository gehostete Rolle der Wert `hg`. Wenn die Rolle auf Ansible Galaxy oder als tar-Archiv auf einem Webserver gehostet wird, können Sie das Keyword `scm` weglassen.

Mit dem Keyword `name` wird der lokale Name der Rolle überschrieben. Mit dem Keyword `version` wird die Version einer Rolle angegeben. Das Keyword `version` kann ein beliebiger Wert sein, der einem Zweig, einem Tag oder einem Commit-Hash aus dem Software-Repository der Rolle entspricht.

## Kapitel 7 | Playbooks mit Rollen vereinfachen

Um die Rollen zu installieren, die einem Playbook-Projekt zugeordnet sind, führen Sie den Befehl `ansible-galaxy install` aus:

```
[user@host project]$ ansible-galaxy install -r roles/requirements.yml \
> -p roles
- downloading role 'redis', owned by geerlingguy
- downloading role from https://github.com/geerlingguy/ansible-role-redis/
archive/1.6.0.tar.gz
- extracting geerlingguy.redis to /opt/project/roles/geerlingguy.redis
- geerlingguy.redis (1.6.0) was installed successfully
- downloading role 'redis', owned by geerlingguy
- downloading role from https://github.com/geerlingguy/ansible-role-redis/
archive/1.5.0.tar.gz
- extracting redis_prod to /opt/project/roles/redis_prod
- redis_prod (1.5.0) was installed successfully
- extracting nginx-acme to /opt/project/roles/nginx-acme
- nginx-acme (56e00a54) was installed successfully
- extracting nginx-acme-ssh to /opt/project/roles/nginx-acme-ssh
- nginx-acme-ssh (master) was installed successfully
- downloading role from file:///opt/local/roles/myrole.tar
- extracting myrole to /opt/project/roles/myrole
- myrole was installed successfully
```

## Verwalten heruntergeladener Rollen

Mit dem Befehl `ansible-galaxy` können auch lokale Rollen, z. B. Rollen aus dem Verzeichnis `roles` eines Playbook-Projekts, verwaltet werden. Der Unterbefehl `ansible-galaxy list` listet die Rollen auf, die lokal gefunden werden.

```
[user@host project]$ ansible-galaxy list
- geerlingguy.redis, 1.6.0
- myrole, (unknown version)
- nginx-acme, 56e00a54
- nginx-acme-ssh, master
- redis_prod, 1.5.0
```

Eine Rolle kann lokal mit dem Unterbefehl `ansible-galaxy remove` entfernt werden.

```
[user@host ~]$ ansible-galaxy remove nginx-acme-ssh
- successfully removed nginx-acme-ssh
[user@host ~]$ ansible-galaxy list
- geerlingguy.redis, 1.6.0
- myrole, (unknown version)
- nginx-acme, 56e00a54
- redis_prod, 1.5.0
```

Verwenden Sie heruntergeladene und installierte Rollen in Playbooks wie andere Rollen. Sie können im Abschnitt `roles` anhand ihrer heruntergeladenen Rollennamen referenziert werden. Wenn eine Rolle nicht im Verzeichnis `roles` des Projekts enthalten ist, wird `roles_path` überprüft, um festzustellen, ob die Rolle in einem dieser Verzeichnisse installiert ist, wobei die erste Übereinstimmung verwendet wird. Das folgende Playbook `use-role.yml` referenziert die Rollen `redis_prod` und `geerlingguy.redis`:

```
[user@host project]$ cat use-role.yml
---
- name: use redis_prod for Prod machines
  hosts: redis_prod_servers
  remote_user: devops
  become: true
  roles:
    - redis_prod

- name: use geerlingguy.redis for Dev machines
  hosts: redis_dev_servers
  remote_user: devops
  become: true
  roles:
    - geerlingguy.redis
```

Dieses Playbook verursacht, dass verschiedene Versionen der Rolle `geerlingguy.redis` für die Produktions- und Entwicklungsserver angewendet werden. Auf diese Weise können Änderungen an der Rolle systematisch getestet und integriert werden, bevor sie auf den Produktionsservern bereitgestellt werden. Durch die Verwendung der Versionskontrolle zur Entwicklung von Rollen können Sie zu einer vorherigen stabilen Version der Rolle zurückkehren, wenn eine kürzlich vorgenommene Änderung einer Rolle Probleme verursacht.



### Literaturhinweise

#### Ansible Galaxy – Ansible-Dokumentation

<https://docs.ansible.com/ansible/2.9/cli/ansible-galaxy.html>

## ► Angeleitete Übung

# Bereitstellen von Rollen mit Ansible Galaxy

In dieser Übung verwenden Sie Ansible Galaxy zum Herunterladen und Installieren einer Ansible-Rolle.

## Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Erstellen einer Rollendatei, um Rollenabhängigkeiten für ein Playbook anzugeben
- Installieren von Rollen, die in einer Rollendatei angegeben sind
- Auflisten von Rollen mit dem Befehl `ansible-galaxy`

## Übersicht über das Szenario

Ihr Unternehmen speichert benutzerdefinierte Dateien auf allen Hosts im Verzeichnis `/etc/skel`. Daher werden neue Benutzerkonten mit einer standardisierten unternehmensspezifischen Bash-Umgebung konfiguriert.

Sie testen die Entwicklungsversion der Ansible-Rolle, die für die Bereitstellung von Gerüstdateien für die Bash-Umgebung verantwortlich ist.

## Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab role-galaxy start` aus. Damit wird das Arbeitsverzeichnis `/home/student/role-galaxy` erstellt und darin eine Ansible-Konfigurationsdatei und das Hostinventar gespeichert.

```
[student@workstation ~]$ lab role-galaxy start
```

## Anweisungen

- 1. Wechseln Sie in das Arbeitsverzeichnis `role-galaxy`.

```
[student@workstation ~]$ cd ~/role-galaxy  
[student@workstation role-galaxy]$
```

- 2. Fügen Sie die Rollenspezifikation einer Rollendatei hinzu, um die Ansible-Rolle zu testen, die Gerüstdateien konfiguriert.

Öffnen Sie Ihren bevorzugten Texteditor und erstellen Sie im Unterverzeichnis `requirements.yml` eine Datei mit dem Namen `roles`. Die URL des Git-Repositorys der Rolle lautet: `git@workstation.lab.example.com:student/bash_env`. Verwenden Sie den `master`-Zweig des Repositorys, um festzustellen, wie sich die Rolle auf das

## Kapitel 7 | Playbooks mit Rollen vereinfachen

Verhalten von Produktionshosts auswirkt. Legen Sie den lokalen Namen der Rolle auf `student.bash_env` fest.

Die `roles/requirements.yml` enthält nun den folgenden Inhalt:

```
---  
# requirements.yml  
  
- src: git@workstation.lab.example.com:student/bash_env  
  scm: git  
  version: master  
  name: student.bash_env
```

- 3. Verwenden Sie den Befehl `ansible-galaxy`, um die gerade erstellte Rollendatei zu verarbeiten und die Rolle `student.bash_env` zu installieren.

- 3.1. Zeigen Sie zum Vergleich den Inhalt des Unterverzeichnisses `roles` an, bevor die Rolle installiert wird.

```
[student@workstation role-galaxy]$ ls roles/  
requirements.yml
```

- 3.2. Verwenden Sie Ansible Galaxy, um die in der Datei `roles/requirements.yml` aufgeführten Rollen herunterzuladen und zu installieren. Stellen Sie sicher, dass alle heruntergeladenen Rollen im Unterverzeichnis `roles` gespeichert werden.

```
[student@workstation role-galaxy]$ ansible-galaxy install -r \  
> roles/requirements.yml -p roles  
- extracting student.bash_env to /home/student/role-galaxy/roles/student.bash_env  
- student.bash_env (master) was installed successfully
```

- 3.3. Zeigen Sie das Unterverzeichnis `roles` an, nachdem die Rolle installiert ist. Überprüfen Sie, ob ein neues Unterverzeichnis mit dem Namen `student.bash_env` vorhanden ist, das mit dem Wert `name` aus der YAML-Datei übereinstimmt.

```
[student@workstation role-galaxy]$ ls roles/  
requirements.yml  student.bash_env
```

- 3.4. Versuchen Sie mit dem Befehl `ansible-galaxy` ohne Optionen die Projektrollen aufzulisten:

```
[student@workstation role-galaxy]$ ansible-galaxy list  
# /usr/share/ansible/roles  
...output omitted...  
- rhel-system-roles.postfix, (unknown version)  
- rhel-system-roles.selinux, (unknown version)  
- rhel-system-roles.timesync, (unknown version)  
# /etc/ansible/roles  
[WARNING]: - the configured path /home/student/.ansible/roles does not exist.
```

Weil Sie die Option `-p` mit dem Befehl `ansible-galaxy install` verwendet haben, wurde die Rolle `student.bash_env` nicht am Standardspeicherort installiert.

## Kapitel 7 | Playbooks mit Rollen vereinfachen

Verwenden Sie die Option -p mit dem Befehl `ansible-galaxy list` zum Auflisten der heruntergeladenen Rollen:

```
[student@workstation role-galaxy]$ ansible-galaxy list -p roles
# /home/student/role-galaxy/roles
- student.bash_env, master
...output omitted...
[WARNING]: - the configured path /home/student/.ansible/roles does not exist.
```



### Anmerkung

Das Verzeichnis `/home/student/.ansible/roles` ist in Ihrem Standard-`roles_path` enthalten, da Sie aber nicht versucht haben, eine Rolle ohne die Option -p zu installieren, hat `ansible-galaxy` das Verzeichnis noch nicht erstellt.

- 4. Erstellen Sie ein Playbook mit dem Namen `use-bash_env-role.yml`, das die Rolle `student.bash_env` verwendet. Der Inhalt des Playbooks sollten mit Folgendem übereinstimmen:

```
---
- name: use student.bash_env role playbook
  hosts: devservers
  vars:
    default_prompt: '[\u on \h in \W dir]\$ '
  pre_tasks:
    - name: Ensure test user does not exist
      user:
        name: student2
        state: absent
        force: yes
        remove: yes

  roles:
    - student.bash_env

  post_tasks:
    - name: Create the test user
      user:
        name: student2
        state: present
        password: "{{ 'redhat' | password_hash('sha512', 'mysecretsalt') }}"
```

Um die Auswirkungen der Konfigurationsänderung zu sehen, muss ein neues Benutzerkonto erstellt werden. Die Abschnitte `pre_tasks` und `post_tasks` des Playbooks stellen sicher, dass das Benutzerkonto `student2` bei jeder Ausführung des Playbooks erstellt wird.

Nach der Ausführung des Playbooks erfolgt der Zugriff auf das Konto `student2` mit dem Passwort `redhat`.

**Anmerkung**

Das `user2`-Passwort wird mit einem Filter generiert. Filter verarbeiten Daten und ändern sie. Hier wird die Zeichenfolge `redhat` geändert, indem sie an das `password_hash`-Modul übergeben wird. Filter sind ein fortgeschrittenes Thema, das in diesem Kurs nicht behandelt wird.

- 5. Führen Sie das Playbook aus. Die Rolle `student.bash_env` erstellt Konfigurationsdateien als Standardvorlage in `/etc/skel` auf dem verwalteten Host. Zu den erstellten Dateien gehören `.bashrc`, `.bash_profile` und `.vimrc`.

```
[student@workstation role-galaxy]$ ansible-playbook use-bash_env-role.yml

PLAY [use student.bash_env role playbook] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Ensure test user does not exist] ****
ok: [servera.lab.example.com]

TASK [student.bash_env : put away .bashrc] ****
changed: [servera.lab.example.com]

TASK [student.bash_env : put away .bash_profile] ****
ok: [servera.lab.example.com]

TASK [student.bash_env : put away .vimrc] ****
changed: [servera.lab.example.com]

TASK [Create the test user] ****
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com    : ok=6      changed=4      unreachable=0      failed=0
```

- 6. Stellen Sie als Benutzer `student2` eine Verbindung mit `servera` über SSH her. Beachten Sie die benutzerdefinierte Aufforderung für den Benutzer `student2` und trennen Sie dann die Verbindung mit `servera`.

```
[student@workstation role-galaxy]$ ssh student2@servera
Activate the web console with: systemctl enable --now cockpit.socket

[student2 on servera in ~ dir]$ exit
logout
Connection to servera closed.
[student@workstation role-galaxy]$
```

**Kapitel 7 |** Playbooks mit Rollen vereinfachen

- 7. Führen Sie das Playbook mit der Entwicklungsversion der Rolle `student.bash_env` aus.

Die Entwicklungsversion der Rolle befindet sich im `dev`-Zweig des Git-Repositories. Die Entwicklungsversion der Rolle verwendet eine neue Variable, `prompt_color`. Bevor Sie das Playbook ausführen, fügen Sie dem `vars`-Abschnitt des Playbooks die Variable `prompt_color` hinzu und legen ihren Wert auf `blue` fest.

- 7.1. Aktualisieren Sie die Datei `roles/requirements.yml`, und legen Sie den Wert `version` auf `dev` fest. Die Datei `roles/requirements.yml` enthält nun:

```
---
# requirements.yml

- src: git@workstation.lab.example.com:student/bash_env
  scm: git
  version: dev
  name: student.bash_env
```

- 7.2. Installieren Sie mit dem Befehl `ansible-galaxy install` die Rolle unter Verwendung der aktualisierten Rollendatei. Überschreiben Sie mit der Option `--force` die vorhandene `master`-Version der Rolle mit der Version `dev` der Rolle.

```
[student@workstation role-galaxy]$ ansible-galaxy install \
> -r roles/requirements.yml --force -p roles
- changing role student.bash_env from master to dev
- extracting student.bash_env to /home/student/role-galaxy/roles/student.bash_env
- student.bash_env (dev) was installed successfully
```

- 7.3. Bearbeiten Sie die Datei `use-bash_env-role.yml`. Fügen Sie dem Abschnitt `vars` des Playbooks die Variable `prompt_color` mit dem Wert `blue` hinzu. Die Datei enthält jetzt Folgendes:

```
---
- name: use student.bash_env role playbook
  hosts: devservers
  vars:
    prompt_color: blue
    default_prompt: '[\u on \h in \W dir]\$ '
  pre_tasks:
...output omitted...
```

- 7.4. Führen Sie das Playbook `use-bash_env-role.yml` aus.

```
[student@workstation role-galaxy]$ ansible-playbook use-bash_env-role.yml

PLAY [use student.bash_env role playbook] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Ensure test user does not exist] ****
changed: [servera.lab.example.com]
```

**Kapitel 7 |** Playbooks mit Rollen vereinfachen

```

TASK [student.bash_env : put away .bashrc] ****
changed: [servera.lab.example.com]

TASK [student.bash_env : put away .bash_profile] ****
changed: [servera.lab.example.com]

TASK [student.bash_env : put away .vimrc] ****
okay: [servera.lab.example.com]

TASK [Create the test user] ****
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com      : ok=6      changed=4      unreachable=0      failed=0

```

- 8. Stellen Sie erneut als `student2` eine Verbindung mit `servera` über SSH her. Beachten Sie den Fehler für den Benutzer `student2` und trennen Sie dann die Verbindung mit `servera`.

```

[student@workstation role-galaxy]$ ssh student2@servera
Activate the web console with: systemctl enable --now cockpit.socket

-bash: [: missing `]'
[student2@servera ~]$ exit
logout
Connection to servera closed.
[student@workstation role-galaxy]$

```

Bei der Analyse der Datei `.bash_profile` des Benutzers `student2` ist ein Bash-Fehler aufgetreten.

- 9. Korrigieren Sie den Fehler in der Entwicklungsversion der Rolle `student.bash_env` und führen Sie das Playbook erneut aus.
- 9.1. Bearbeiten Sie die Datei `roles/student.bash_env/templates/_bash_profile.j2`. Fügen Sie das fehlende `]`-Zeichen in Zeile 4 ein und speichern Sie die Datei. Der Anfang der Datei sieht jetzt folgendermaßen aus:

```

# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/.local/bin:$HOME/bin

export PATH

```

Speichern Sie die Datei.

- 9.2. Führen Sie das Playbook `use-bash_env-role.yml` aus.

**Kapitel 7 |** Playbooks mit Rollen vereinfachen

```
[student@workstation role-galaxy]$ ansible-playbook use-bash_env-role.yml

PLAY [use student.bash_env role playbook] ****

TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Ensure test user does not exist] ****
changed: [servera.lab.example.com]

TASK [student.bash_env : put away .bashrc] ****
ok: [servera.lab.example.com]

TASK [student.bash_env : put away .bash_profile] ****
changed: [servera.lab.example.com]

TASK [student.bash_env : put away .vimrc] ****
ok: [servera.lab.example.com]

TASK [Create the test user] ****
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com      : ok=6      changed=3      unreachable=0      failed=0
```

9.3. Stellen Sie erneut als `student2` eine Verbindung mit `servera` über SSH her.

```
[student@workstation role-galaxy]$ ssh student2@servera
Activate the web console with: systemctl enable --now cockpit.socket

[student2 on servera in ~ dir]$ exit
logout
Connection to servera closed.
[student@workstation role-galaxy]$
```

Die Fehlermeldung ist nicht mehr vorhanden. Die benutzerdefinierte Aufforderung für den Benutzer `student2` wird jetzt in blauen Zeichen angezeigt.

- **10.** Die obigen Schritte zeigen, dass die Entwicklungsversion der Rolle `student.bash_env` fehlerhaft ist. Basierend auf den Testergebnissen übernehmen die Entwickler die erforderlichen Korrekturen wieder in den Entwicklungszweig der Rolle. Wenn der Entwicklungszweig die erforderlichen Qualitätsprüfungen besteht, führen die Entwickler Funktionen aus dem Entwicklungszweig mit dem Zweig `master` zusammen.  
Das Übergeben von Rollenänderungen an ein Git-Repository wird in diesem Kurs nicht behandelt.



### Wichtig

Wenn Sie die neueste Version einer Rolle in einem Projekt nachverfolgen, installieren Sie die Rolle regelmäßig neu, um sie zu aktualisieren. Dadurch wird sichergestellt, dass die lokale Kopie mit Fehlerbehebungen, Patches und anderen Funktionen auf dem neuesten Stand bleibt.

Wenn Sie jedoch eine Rolle eines Drittanbieters in der Produktion verwenden, sollten Sie die zu verwendende Version angeben, um Probleme durch unerwartete Änderungen zu vermeiden. Führen Sie in der Testumgebung regelmäßig eine Aktualisierung auf die neueste Version der Rolle durch, damit Verbesserungen und Änderungen kontrolliert übernommen werden.

## Beenden

Führen Sie den Befehl `lab role-galaxy finish` aus, um den verwalteten Host zu bereinigen.

```
[student@workstation ~]$ lab role-galaxy finish
```

Hiermit ist die angeleitete Übung beendet.

# Abrufen von Rollen und Modulen aus Content-Sammlungen

---

## Ziele

Am Ende dieses Abschnitts sollten Sie in der Lage sein, eine Reihe von verwandten Rollen, Zusatzmodulen und anderen Inhalten aus Content-Sammlungen abzurufen und sie in einem Playbook zu verwenden.

## Erläutern von Content-Sammlungen

*Ansible-Content-Sammlungen* sind ein Distributionsformat für Ansible-Inhalt. Eine Sammlung bietet eine Reihe von verwandten Modulen, Rollen und Plugins, die Sie auf Ihren Kontrollknoten herunterladen und dann in Ihren Playbooks verwenden können.

Beispiel:

- Die `redhat.insights`-Sammlung enthält Module und Rollen, mit denen Sie ein System bei Red Hat Insights for Red Hat Enterprise Linux registrieren können.
- Die `cisco-ios`-Sammlung enthält Module und Plugins zur Verwaltung von Cisco iOS-Netzwerk-Appliances. Das Unternehmen Cisco unterstützt und verwaltet diese Sammlung.
- Die `community.crypto`-Sammlung bietet Module, die SSL/TLS-Zertifikate erstellen.

Content-Sammlungen ermöglichen es, Aktualisierungen des Ansible-Kerncodes von Aktualisierungen von Modulen und Plugins zu trennen. Dies ermöglicht es Anbietern und Entwicklern, ihre Sammlungen in ihrem eigenen Tempo und unabhängig von Ansible-Versionen zu pflegen und zu verteilen. Sie können Ihre eigenen Sammlungen entwickeln, um Ihren Teams benutzerdefinierte Rollen und Module zur Verfügung zu stellen.

Content-Sammlungen geben Ihnen auch mehr Flexibilität. Sammlungen ermöglichen es Ihnen, nur die Inhalte zu installieren, die Sie benötigen, anstatt alle unterstützten Module installieren zu müssen. Sie können auch eine bestimmte Version einer Sammlung auswählen (vielleicht eine frühere oder spätere) oder zwischen einer von Red Hat oder von Anbietern unterstützten Sammlungsversion oder einer von der Community bereitgestellten Version einer Sammlung wählen.

Ansible 2.9 und höher unterstützt Sammlungen. Spätere Versionen von Ansible und Red Hat Ansible Automation Platform werden weitere Verbesserungen des Sammlungs-Supports enthalten und Sammlungen umfassend nutzen. Es wird wichtig sein, die Funktionsweise von Sammlungen zu verstehen.

Das RPM-Paket `ansible`, das mit Red Hat Ansible Automation Platform 1.2 und Ansible 2.9 bereitgestellt wird, installiert automatisch alle Module, die in früheren Versionen von Ansible installiert wurden. Zukünftige Versionen von Ansible und Red Hat Ansible Automation Platform entfernen die meisten Module aus dem RPM-Hauptpaket und platzieren sie in Sammlungen, die entweder enthalten sind oder heruntergeladen werden müssen.

## Organisieren von Sammlungen in Namespaces

Um die Angabe von Sammlungen und deren Inhalt nach Name zu vereinfachen, werden Sammlungsnamen in *Namespaces* organisiert. Anbieter, Partner, Entwickler und Ersteller von

## Kapitel 7 | Playbooks mit Rollen vereinfachen

Inhalten können Namespaces verwenden, um ihren Sammlungen eindeutige Namen zuzuweisen, ohne mit anderen Entwicklern in Konflikt zu geraten.

Der Namespace ist der erste Teil eines Sammlungsnamens. Beispielsweise befinden sich alle Sammlungen, die die Ansible-Community pflegt, im Community-Namespace, und sie haben Namen wie `community.crypto`, `community.postgresql` und `community.rabbitmq`. Sammlungen, die Red Hat pflegt und unterstützt, können den redhat-Namespace verwenden und Namen wie `redhat.rhv`, `redhat.satellite` und `redhat.insights` aufweisen.

## Auswählen von Sammlungsquellen

Ansible bietet zwei offizielle Quellen zum Herunterladen und Installieren von Sammlungen: Ansible Automation Hub und Ansible Galaxy.

### Ansible Automation Hub

Ansible Automation Hub hostet Ansible-Content-Sammlungen, die Red Hat und seine Partner für ihre Kunden unterstützen. Red Hat überprüft, pflegt, aktualisiert und unterstützt diese Sammlungen vollständig. Beispielsweise sind die Sammlungen `redhat.rhv`, `redhat.satellite`, `redhat.insights` und `cisco.ios` auf dieser Plattform verfügbar.

Sie benötigen ein gültiges Abonnement für Red Hat Ansible Automation Platform, um auf den Ansible Automation Hub zugreifen zu können. Verwenden Sie die Webbenutzeroberfläche des Ansible Automation Hub unter <https://cloud.redhat.com/ansible/automation-hub/>, um die Sammlungen aufzulisten und aufzurufen.

### Ansible Galaxy

Ansible Galaxy hostet Sammlungen, die von einer Vielzahl von Ansible-Entwicklern und - Benutzern eingereicht wurden. Ansible Galaxy ist eine öffentliche Bibliothek ohne offizielle Support-Garantien, die jedoch öffentlichen Zugriff ermöglicht. Beispielsweise sind die Sammlungen `community.crypto`, `community.postgresql` und `community.rabbitmq` über diese Plattform verfügbar.

Verwenden Sie die Webbenutzeroberfläche von Ansible Galaxy unter <https://galaxy.ansible.com/>, um nach Sammlungen zu suchen.

## Installieren von Content-Sammlungen

Bevor Ihre Playbooks Inhalte aus einer Sammlung verwenden können, müssen Sie die entsprechende Sammlung auf Ihrem Kontrollknoten installieren. Verwenden Sie den Befehl `ansible-galaxy`, um Sammlungen aus verschiedenen Quellen, darunter Ansible Galaxy, herunterzuladen.

Im folgenden Beispiel wird der Befehl `ansible-galaxy` mit dem Argument `collection` verwendet, um die Sammlung `community.crypto` herunterzuladen und anschließend auf dem lokalen System zu installieren.

```
[user@controlnode ~]$ ansible-galaxy collection install community.crypto
```

Mit dem Befehl kann auch eine Sammlung aus einem lokalen oder einem Remote-TAR-Archiv installiert werden.

```
[user@controlnode ~]$ ansible-galaxy collection install \
> /tmp/community-dns-1.2.0.tar.gz
[user@controlnode ~]$ ansible-galaxy collection install \
> http://www.example.com/redhat-insights-1.0.5.tar.gz
```

## Kapitel 7 | Playbooks mit Rollen vereinfachen

Die Ansible -Konfigurationsanweisung `collections_paths` gibt eine durch Doppelpunkte getrennte Liste von Pfaden im System an, in denen Ansible nach installierten Sammlungen sucht. Sie können diese Anweisung in der Konfigurationsdatei `ansible.cfg` festlegen. Standardmäßig installiert der Befehl `ansible-galaxy` die Sammlungen im ersten Verzeichnis, das die Anweisung `collections_paths` definiert.

Der Standardwert für `collections_paths` ist `~/.ansible/collections:/usr/share/ansible/collections`. Daher installiert der Befehl `ansible-galaxy` Sammlungen standardmäßig im Verzeichnis `~/.ansible/collections`.

Wenn Sie eine Sammlung in einem anderen Verzeichnis installieren möchten, verwenden Sie die Option `--collections-path` (oder `-p`).

```
[root@controlnode ~]# ansible-galaxy collection install \
> -p /usr/share/ansible/collections community.postgresql
```



### Wichtig

Wenn Sie die Option `--collections-path` (oder `-p`) verwenden, müssen Sie ein Verzeichnis auswählen, das in der Anweisung `collections_paths` aufgeführt ist. Der Befehl `ansible-playbook` verwendet diese Anweisung auch, um Sammlungen zu finden. Wenn Sie keinen Pfad verwenden, der in der Anweisung `collections_paths` definiert ist, finden Ihre Playbooks die von Ihnen installierten Sammlungen nicht.

## Installieren von Sammlungen mit einer Anforderungsdatei

Sie können eine `requirements.yml`-Datei erstellen, um alle Sammlungen aufzulisten, die Sie installieren müssen. Durch das Hinzufügen einer `collections/requirements.yml`-Datei in Ihrem Ansible-Projekt können Ihre Teammitglieder sofort die erforderlichen Sammlungen identifizieren. Außerdem erkennt der Automation Controller diese Datei und installiert die Sammlungen automatisch, bevor Ihre Playbooks ausgeführt werden.

Die folgende `requirements.yml`-Datei listet mehrere zu installierende Sammlungen auf. Beachten Sie, dass Sie eine bestimmte Sammlungsversion anvisieren und auch lokale oder Remote-Tar-Archive bereitstellen können.

```
---
collections:
  - name: community.crypto

  - name: ansible.posix
    version: 1.2.0

  - name: /tmp/community-dns-1.2.0.tar.gz

  - name: http://www.example.com/redhat-insights-1.0.5.tar.gz
```

Der Befehl `ansible-galaxy` kann dann diese Datei verwenden, um alle diese Sammlungen zu installieren. Verwenden Sie die Option `--requirements-file` (oder `-r`), um dem Befehl die Datei `requirements.yml` zur Verfügung zu stellen.

```
[root@controlnode ~]# ansible-galaxy collection install -r requirements.yml
```

## Konfigurieren von Sammlungsquellen

Standardmäßig verwendet der Befehl `ansible-galaxy` Ansible Galaxy unter `https://galaxy.ansible.com/`, um Sammlungen herunterzuladen.

Damit der Befehl auch Ansible Automation Hub verwendet, fügen Sie der Datei `ansible.cfg` die folgenden Anweisungen hinzu.

```
...output omitted...
[galaxy]
server_list = automation_hub, galaxy ①

[galaxy_server.automation_hub]
url=https://cloud.redhat.com/api/automation-hub/ ②
auth_url=https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-
connect/token ③
token=eyJh...Jf0o ④

[galaxy_server.galaxy]
url=https://galaxy.ansible.com/
```

- ① Listen Sie alle Repositorys, die der Befehl `ansible-galaxy` verwenden soll, in der gewünschten Reihenfolge auf. Fügen Sie für jeden von Ihnen definierten Namen einen Abschnitt `[galaxy_server.name]` hinzu, um die Verbindungsparameter anzugeben. Da Ansible Automation Hub möglicherweise nicht alle Sammlungen bietet, die Ihre Playbooks benötigen, können Sie Ansible Galaxy an letzter Stelle als Fallback hinzufügen. Wenn die Sammlung nicht im Ansible Automation Hub verfügbar ist, verwendet der Befehl `ansible-galaxy` dann Ansible Galaxy, um sie abzurufen.
- ② Geben Sie die URL für den Zugriff auf das Repository an.
- ③ Geben Sie die URL für die Authentifizierung an.
- ④ Um auf den Ansible Automation Hub zugreifen zu können, benötigen Sie ein Authentifizierungstoken, das Ihrem Konto zugeordnet ist. Verwenden Sie die Webbenutzeroberfläche des Ansible Automation Hub, um dieses Token zu generieren. Weitere Informationen zu diesem Vorgang finden Sie unter den Links im Abschnitt „Referenzen“.

Anstelle eines Tokens können Sie die Parameter `username` und `password` verwenden, um Benutzernamen und Passwort für das Kundenportal anzugeben.

```
...output omitted...
[galaxy_server.automation_hub]
url=https://cloud.redhat.com/api/automation-hub/
username=operator1
password=Sup3r53cR3t
...output omitted...
```

Vielleicht möchten Sie Ihre Anmeldeinformationen nicht in der Datei `ansible.cfg` offenlegen, da die Datei über die Versionskontrolle übergeben werden könnte. Entfernen Sie in diesem Fall die Authentifizierungsparameter aus der Datei `ansible.cfg`, und definieren Sie sie als Umgebungsvariablen. Sie definieren die Umgebungsvariablen wie folgt:

## Kapitel 7 | Playbooks mit Rollen vereinfachen

```
ANSIBLE_GALAXY_SERVER_<server_id>_<key>=value
```

### server\_id

Serverkennung in Großbuchstaben. Die Serverkennung ist der Name, den Sie im Parameter `server_list` und im Abschnitt `[galaxy_server.server_id]` verwenden.

### key

Name des Parameters in Großbuchstaben.

Im folgenden Beispiel wird der Parameter `token` als Umgebungsvariable bereitgestellt:

```
[user@controlnode ~]$ cat ansible.cfg
...output omitted...
[galaxy_server.automation_hub]
url=https://cloud.redhat.com/api/automation-hub/
auth_url=https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-
connect/token
[user@controlnode ~]$ export \
> ANSIBLE_GALAXY_SERVER_AUTOMATION_HUB_TOKEN='eyJh...Jf0o'
[user@controlnode ~]$ ansible-galaxy collection install ansible.posix
```

## Verwenden von Sammlungen

Nachdem Sie eine Sammlung installiert haben, können Sie sie mit Ad-hoc-Befehlen und Playbooks verwenden. In der Sammlungsdokumentation im Ansible Automation Hub oder in der Webbenutzeroberfläche von Ansible Galaxy finden Sie Informationen über die bereitgestellten Rollen und Module. Alternativ können Sie die Verzeichnisstruktur der Sammlungen in Ihrem System überprüfen. Die Sammlung speichert die Module im Verzeichnis `plugins/modules/` und die Rollen im Verzeichnis `roles/`.

```
[user@controlnode ~]$ tree \
> ~/.ansible/collections/ansible_collections/redhat/insights/
/home/user/.ansible/collections/ansible_collections/redhat/insights/
...output omitted...
├── plugins
│   ├── action
│   │   └── insights_config.py
│   ├── inventory
│   │   └── insights.py
│   └── modules
│       ├── insights_config.py
│       └── insights_register.py
...output omitted...
└── roles
    ├── compliance
    │   ├── meta
    │   │   └── main.yml
    │   ├── README.md
    │   ├── tasks
    │   │   └── install.yml
    │   ├── └── main.yml
    │   └── └── run.yml
    └── tests
```

**Kapitel 7 |** Playbooks mit Rollen vereinfachen

```
|   |   └── compliance.yml  
|   |   └── install-only.yml  
|   |   └── run-only.yml  
|   └── insights_client  
...output omitted...
```

Um ein Modul oder eine Rolle zu verwenden, verweisen Sie mit dem *voll qualifizierten Sammlungsnamen* (FQCN) darauf. Basierend auf der vorherigen Ausgabe verweisen Sie auf die Rolle `insights_client` mit `redhat.insights.insights_client`.

Mit dem folgenden Ad-hoc-Befehl wird das Modul `mail` aus der Sammlung `community.general` aufgerufen.

```
[user@controlnode ~]$ ansible localhost -m community.general.mail \  
> -a 'subject="Hello World" to=root'
```

Das folgende Playbook ruft das Modul `mysql_user` aus der Sammlung `community.mysql` auf.

```
---  
- name: Create the operator1 user in the test database  
  hosts: db.example.com  
  
  tasks:  
    - name: Ensure the operator1 database user is defined  
      community.mysql.mysql_user:  
        name: operator1  
        password: Secret0451  
        priv: '.:ALL'  
        state: present
```

Das folgende Playbook verwendet die Rolle `organizations` aus der Sammlung `redhat.satellite`.

```
---  
- name: Add the test organizations to Red Hat Satellite  
  hosts: localhost  
  
  tasks:  
    - name: Ensure the organizations exist  
      include_role:  
        name: redhat.satellite.organizations  
    vars:  
      satellite_server_url: https://sat.example.com  
      satellite_username: admin  
      satellite_password: Sup3r53cr3t  
      satellite_organizations:  
        - name: test1  
          label: tst1  
          state: present  
          description: Test organization 1  
        - name: test2
```

```
label: tst2
state: present
description: Test organization 2
```

## Verwenden von integrierten Ansible-Sammlungen nach Ansible 2.9

In zukünftigen Versionen von Ansible wird die Kerninstallation immer eine spezielle Sammlung namens `ansible.builtin` enthalten. Diese Sammlung umfasst eine Reihe von gängigen Modulen wie `copy`, `template`, `file`, `yum`, `command` und `service`.

Sie können immer die Kurznamen dieser Module in Ihren Playbooks verwenden. Zum Beispiel können Sie weiterhin `file` anstelle von `ansible.builtin.file` verwenden. Dadurch können viele Ansible 2.9-Playbooks ohne Änderungen verwendet werden. Sie müssen aber möglicherweise zusätzliche Sammlungen für Module installieren, die nicht in `ansible.builtin` enthalten sind.

Red Hat empfiehlt jedoch, die FQCN-Notation zu verwenden, um zukünftige Konflikte mit Sammlungen zu vermeiden, die dieselben Modulnamen verwenden.

Das folgende Playbook verwendet die FQCN-Notation für die Module `yum`, `copy` und `service`.

```
---
- name: Install and start Apache HTTPD
  hosts: web

  tasks:
    - name: Ensure the httpd package is present
      ansible.builtin.yum:
        name: httpd
        state: present

    - name: Ensure the index.html file is present
      ansible.builtin.copy:
        src: files/index.html
        dest: /var/www/html/index.html
        owner: root
        group: root
        mode: 0644
        setype: httpd_sys_content_t

    - name: Ensure the httpd service is started
      ansible.builtin.service:
        name: httpd
        state: started
        enabled: true
```



### Literaturhinweise

#### **Using collections – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/user\\_guide/collections\\_using.html](https://docs.ansible.com/ansible/2.9/user_guide/collections_using.html)

#### **Galaxy User Guide – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/galaxy/user\\_guide.html](https://docs.ansible.com/ansible/2.9/galaxy/user_guide.html)

## ► Angeleitete Übung

# Abrufen von Rollen und Modulen aus Content-Sammlungen

In dieser Übung installieren Sie eine Content-Sammlung und verwenden eine Rolle oder ein Modul aus dieser Content-Sammlung in einem Playbook.

## Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Verwenden des Befehls `ansible-galaxy`, um eine Content-Sammlung zu installieren.
- Verwenden einer `requirements.yml`-Datei, um mehrere Sammlungen zu installieren.
- Aufrufen der Rollen und Module der Content-Sammlungen aus Playbooks.

## Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab role-collections start` aus. Mit diesem Befehl wird das Arbeitsverzeichnis `/home/student/role-collections` erstellt, und darin wird ein Ansible-Projekt gespeichert.

```
[student@workstation ~]$ lab role-collections start
```

## Anweisungen

### ► 1. Installieren und überprüfen Sie dann die Sammlung `gls.utils`.

- 1.1. Installieren Sie die Sammlung `gls.utils` aus der TAR-Datei unter <http://materials.example.com/labs/role-collections/gls-utils-0.0.1.tar.gz>. Sie können diese URL aus der Datei `/home/student/role-collections/url.txt` kopieren und einfügen.

```
[student@workstation ~]$ ansible-galaxy collection install \
> http://materials.example.com/labs/role-collections/gls-utils-0.0.1.tar.gz
Process install dependency map
Starting collection install process
Installing 'gls.utils:0.0.1' to '/home/student/.ansible/collections/
ansible_collections/gls/utils'
```

Beachten Sie, dass mit dem vorherigen Befehl die Sammlung im Verzeichnis `/home/student/.ansible/collections/ansible_collections/gls/utils` installiert wird.

- 1.2. Listen Sie die Rollen auf, die die Sammlung bereitstellt.

```
[student@workstation ~]$ ls \
> ~/.ansible/collections/ansible_collections/gls/utils/roles
backup  restore
```

In der vorherigen Ausgabe sehen Sie, dass die Sammlung zwei Rollen bietet: `backup` und `restore`.

- 1.3. Jede Rolle stellt eine `README.md`-Datei bereit. Sehen Sie sich die `README.md`-Datei für die Rolle `backup` an.

```
[student@workstation ~]$ cat \
> ~/.ansible/collections/ansible_collections/gls/utils/roles/backup/README.md
...output omitted...
```

- 1.4. Listen Sie die Module auf, die die Sammlung bereitstellt.

```
[student@workstation ~]$ ls \
> ~/.ansible/collections/ansible_collections/gls/utils/plugins/modules
newping.py
```

Die Sammlung enthält das Modul `newping`.

- 1.5. Verwenden Sie den Befehl `ansible-doc`, um die Dokumentation für das Modul `newping` abzurufen.

```
[student@workstation ~]$ ansible-doc gls.utils.newping
...output omitted...
```

- 2. Schließen Sie den Vorgang ab, und führen Sie anschließend das Playbook `home/student/role-collections/bck.yml` aus. Dieses Playbook verwendet das Modul `gls.utils.newping` und die Rolle `gls.utils.backup`.

- 2.1. Wechseln Sie in das Arbeitsverzeichnis `role-collections`.

```
[student@workstation ~]$ cd ~/role-collections
[student@workstation role-collections]$
```

- 2.2. Bearbeiten Sie das Playbook `bck.yml`. Rufen Sie in der ersten Aufgabe das Modul `gls.utils.newping` auf.

```
...output omitted...
tasks:
  - name: Ensure the machine is up
    gls.utils.newping:
      data: pong
...output omitted...
```

Schließen Sie die Datei noch nicht.

- 2.3. Rufen Sie in der zweiten Aufgabe die Rolle `gls.utils.backup` auf. Speichern und schließen Sie anschließend die Datei.

## Kapitel 7 | Playbooks mit Rollen vereinfachen

```
...output omitted...
- name: Ensure configuration files are saved
  include_role:
    name: gls.utils.backup
  vars:
    backup_id: backup_etc
    backup_files:
      - /etc/sysconfig
      - /etc/yum.repos.d
...output omitted...
```

Die resultierende Datei sollte wie folgt angezeigt werden:

```
---
- name: Backup the system configuration
  hosts: servera.lab.example.com
  become: true
  gather_facts: false

  tasks:
    - name: Ensure the machine is up
      gls.utils.newping:
        data: pong

    - name: Ensure configuration files are saved
      include_role:
        name: gls.utils.backup
      vars:
        backup_id: backup_etc
        backup_files:
          - /etc/sysconfig
          - /etc/yum.repos.d
```

2.4. Verifizieren Sie die Syntax des Playbooks bck.yml.

```
[student@workstation role-collections]$ ansible-playbook --syntax-check bck.yml
playbook: bck.yml
```

2.5. Führen Sie das Playbook aus.

```
[student@workstation role-collections]$ ansible-playbook bck.yml
...output omitted...
```

- 3. Installieren Sie im zweiten Teil dieser Übung Content-Sammlungen, die durch eine requirements.yml-Datei spezifiziert werden.

Führen Sie das Playbook new\_system.yml aus, um Ihre Arbeit zu testen, wenn Sie fertig sind. Dieses Playbook verwendet die Rollen redhat.insights.insights\_client und redhat.rhel\_system\_roles.selinux, um Red Hat Insights und SELinux auf dem servera-Rechner zu konfigurieren.

**Kapitel 7 |** Playbooks mit Rollen vereinfachen

- 3.1. Überprüfen Sie die Datei `requirements.yml`. In der Datei werden zwei aus TAR-Dateien zu installierende Sammlungen aufgelistet, die auf dem Webserver `materials.example.com` gehostet werden.

```
---  
collections:  
  - name: http://materials.example.com/labs/role-collections/redhat-  
    insights-1.0.5.tar.gz  
  - name: http://materials.example.com/labs/role-collections/redhat-  
    rhel_system_roles-1.0.1.tar.gz
```

- 3.2. Verwenden Sie den Befehl `ansible-galaxy` mit der Datei `requirements.yml`, um die Sammlungen zu installieren.

```
[student@workstation role-collections]$ ansible-galaxy collection install \  
> -r requirements.yml  
Process install dependency map  
Starting collection install process  
Installing 'redhat.insights:1.0.5' to '/home/student/.ansible/collections/  
ansible_collections/redhat/insights'  
Installing 'redhat.rhel_system_roles:1.0.1' to '/home/student/.ansible/  
collections/ansible_collections/redhat/rhel_system_roles'
```

- 3.3. Prüfen Sie das Playbook `new_system.yml`.

```
---  
- name: Configure the system  
  hosts: servera.lab.example.com  
  become: true  
  gather_facts: true  
  
  tasks:  
    - name: Ensure the system is registered with Insights  
      include_role:  
        name: redhat.insights.insights_client  
      vars:  
        auto_config: false  
        insights_proxy: http://proxy.example.com:8080  
  
    - name: Ensure SELinux mode is Enforcing  
      include_role:  
        name: redhat.rhel_system_roles.selinux  
      vars:  
        selinux_state: enforcing
```

- 3.4. Führen Sie das Playbook `new_system.yml` im Check-Modus aus, um zu überprüfen, ob die erforderlichen Sammlungen korrekt installiert wurden. Da die Kursraumsysteme nicht bei Red Hat registriert sind und möglicherweise keinen Internetzugang haben, kann das Playbook `new_system.yml` nicht erfolgreich abgeschlossen werden. Sie können das Playbook zur Überprüfung der korrekten Installation der erforderlichen Sammlungen jedoch im Check-Modus ausführen.

```
[student@workstation role-collections]$ ansible-playbook --check new_system.yml
```

## Beenden

Führen Sie den Befehl `lab role-collections finish` aus, um den verwalteten Host zu bereinigen.

```
[student@workstation ~]$ lab role-collections finish
```

Hiermit ist die angeleitete Übung beendet.

## ► Praktische Übung

# Vereinfachen von Playbooks mit Rollen

### Performance-Checkliste

In dieser praktischen Übung erstellen Sie Ansible-Rollen, die Variablen, Dateien, Vorlagen, Aufgaben und Handler verwenden.

### Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Erstellen von Ansible-Rollen, die zum Konfigurieren eines Entwicklungs-Webservers Variablen, Dateien, Vorlagen, Aufgaben und Handler verwenden
- Verwenden einer Rolle, die in einem Remote-Repository in einem Playbook gehostet wird
- Verwenden einer Red Hat Enterprise Linux System Role in einem Playbook

### Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab role-review start` aus. Das Skript erstellt das Arbeitsverzeichnis `/home/student/role-review` und füllt es mit einer Ansible-Konfigurationsdatei, dem Hostinventar und anderen Lab-Dateien.

```
[student@workstation ~]$ lab role-review start
```

### Anweisungen

Ihr Unternehmen muss einen einzelnen Webserver bereitstellen, um Entwicklungscode für alle Webentwickler zu hosten. Sie sollen ein Playbook zur Konfiguration dieses Entwicklungs-Webservers schreiben.

Der Entwicklungs-Webserver muss mehrere Anforderungen erfüllen:

- Die Konfiguration des Entwicklungsservers stimmt mit der Konfiguration des Produktionsservers überein. Der Produktionsserver ist mit einer Ansible-Rolle konfiguriert, die vom Infrastruktur-Team des Unternehmens entwickelt wurde.
- Jeder Entwickler erhält auf dem Entwicklungsserver ein Verzeichnis, in dem er Code und Inhalt ablegt. Auf den Inhalt jedes Entwicklers wird über einen zugewiesenen, nicht dem Standard entsprechenden Port zugegriffen.
- SELinux ist auf „enforcing“ und „targeted“ festgelegt.

In Ihrem Playbook soll:

- Eine Rolle verwendet werden, um Verzeichnisse und Ports für jeden Entwickler auf dem Webserver zu konfigurieren. Sie müssen diese Rolle schreiben.

## Kapitel 7 | Playbooks mit Rollen vereinfachen

Diese Rolle hängt von einer Rolle ab, die vom Unternehmen zur Konfiguration von Apache entwickelt wurde. Sie sollen die Abhängigkeit anhand der Version v1.4 der Unternehmensrolle definieren. Die URL des Repository mit der Abhängigkeit lautet:  
`git@workstation.lab.example.com:infra/apache`

- Verwenden Sie die Rolle `rhel-system-roles.selinux` zum Konfigurieren von SELinux für die von Ihrem Webserver verwendeten nicht standardmäßigen HTTP-Ports. Sie erhalten eine `selinux.yml`-Variablendatei, die als `group_vars`-Datei installiert werden kann, um die korrekten Einstellungen an die Rolle zu übergeben.

1. Wechseln Sie in das Arbeitsverzeichnis `/home/student/role-review`.
2. Erstellen Sie ein Playbook mit dem Namen `web_dev_server.yml` mit einem einzelnen Play namens `Configure Dev Web Server`. Konfigurieren Sie das Play so, dass es auf die Hostgruppe `dev_webserver` abzielt. Fügen Sie dem Play noch keine Rollen oder Aufgaben hinzu.

Stellen Sie sicher, dass das Play die Ausführung von Handlern erzwingt, da beim Entwickeln des Playbooks ein Fehler auftreten könnte.

3. Prüfen Sie die Syntax des Playbooks. Führen Sie das Playbook aus. Die Syntaxprüfung sollte bestanden und das Playbook erfolgreich ausgeführt werden.
4. Stellen Sie sicher, dass die Rollenabhängigkeiten des Playbooks installiert sind.

Die Rolle `apache.developer_configs`, die Sie erstellen sollen, hängt von der Rolle `infra.apache` ab. Erstellen Sie eine Datei `roles/requirements.yml`. Sie sollte die Rolle aus dem Git-Repository unter `git@workstation.lab.example.com:infra/apache` installieren, die Version v1.4 verwenden und lokal `infra.apache` benannt werden. Sie können davon ausgehen, dass Ihre SSH-Schlüssel so konfiguriert sind, dass Sie Rollen aus diesem Repository automatisch abrufen können. Installieren Sie die Rolle mit dem Befehl `ansible-galaxy`.

Installieren Sie zusätzlich das Paket `rhel-system-roles`, falls noch nicht vorhanden.

5. Initialisieren Sie im Unterverzeichnis `roles` eine neue Rolle mit dem Namen `apache.developer_configs`. Fügen Sie die Rolle `infra.apache` als Abhängigkeit für die neue Rolle hinzu, und verwenden Sie dafür dieselben Informationen für Name, Quelle, Version und Versionskontrollsystem wie die Datei `roles/requirements.yml`.

Die Datei `developer_tasks.yml` im Projektverzeichnis enthält Aufgaben für die Rolle. Verschieben Sie diese Datei an den richtigen Speicherort, damit sie zur Aufgabendatei für diese Rolle wird, und ersetzen Sie die vorhandene Datei an diesem Speicherort.

Bei der Datei `developer.conf.j2` im Projektverzeichnis handelt es sich um eine Jinja2-Vorlage, die von der Aufgabendatei verwendet wird. Verschieben Sie sie an den richtigen Speicherort für die von dieser Rolle verwendeten Vorlagendateien.

6. Die Rolle `apache.developer_configs` verarbeitet eine Liste von Benutzern, die in einer Variable mit dem Namen `web_developers` definiert sind. In der Datei `web_developers.yml` im Projektverzeichnis ist die Benutzerlistenvariable `web_developers` definiert. Überprüfen Sie diese Datei, und verwenden Sie sie zur Definition der Variable `web_developers` für die Hostgruppe des Entwicklungs-Webservers.
7. Fügen Sie dem Play im Playbook `web_dev_server.yml` die Rolle `apache.developer_configs` hinzu.
8. Prüfen Sie die Syntax des Playbooks. Führen Sie das Playbook aus. Die Syntaxprüfung sollte bestanden werden, das Playbook sollte jedoch fehlschlagen, wenn die Rolle `infra.apache` versucht, Apache HTTPD neu zu starten.

## Kapitel 7 | Playbooks mit Rollen vereinfachen

9. Apache HTTPD konnte im vorherigen Schritt nicht neu gestartet werden, da die für Ihre Entwickler verwendeten Netzwerkports mit den falschen SELinux-Kontexten gekennzeichnet sind. Sie haben die Variablendatei `selinux.yml` erhalten, die mit der Rolle `rhel-system-roles.selinux` verwendet werden kann, um das Problem zu beheben.  
Erstellen Sie im Playbook `web_dev_server.yml` den Abschnitt `pre_tasks` für Ihr Play. Verwenden Sie in diesem Abschnitt eine Aufgabe, um die Rolle `rhel-system-roles.selinux` in eine `block/rescue`-Struktur einzubeziehen, damit sie richtig angewendet wird. Lesen Sie das Kursskript oder die Dokumentation für diese Rolle, um zu erfahren, wie Sie diese Aufgabe durchführen können.  
Überprüfen Sie die Datei `selinux.yml`. Verschieben Sie sie an den richtigen Speicherort, damit die Variablen für die Hostgruppe `dev_webserver` festgelegt werden.
10. Überprüfen Sie das finale Playbook `web_dev_server.yml`, und führen Sie eine Syntaxprüfung durch. Die Syntaxprüfung sollte bestanden werden.  
Überprüfen Sie, ob das Playbook `web_dev_server.yml` eine Syntaxprüfung besteht.
11. Führen Sie das Playbook aus. Es sollte korrekt ausgeführt werden.
12. Testen Sie die Konfiguration des Entwicklungs-Webservers. Stellen Sie sicher, dass auf alle Endpunkte zugegriffen werden kann und sie den Inhalt der einzelnen Entwickler bereitstellen.

## Bewertung

Bewerten Sie Ihre Arbeit, indem Sie auf Ihrem `workstation`-Rechner den Befehl `lab role-review grade` ausführen. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie das Skript so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab role-review grade
```

## Beenden

Führen Sie auf `workstation` das Skript `lab role-review finish` aus, um diese Übung zu bereinigen.

```
[student@workstation ~]$ lab role-review finish
```

Hiermit ist die praktische Übung abgeschlossen.

## ► Lösung

# Vereinfachen von Playbooks mit Rollen

### Performance-Checkliste

In dieser praktischen Übung erstellen Sie Ansible-Rollen, die Variablen, Dateien, Vorlagen, Aufgaben und Handler verwenden.

### Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Erstellen von Ansible-Rollen, die zum Konfigurieren eines Entwicklungs-Webservers Variablen, Dateien, Vorlagen, Aufgaben und Handler verwenden
- Verwenden einer Rolle, die in einem Remote-Repository in einem Playbook gehostet wird
- Verwenden einer Red Hat Enterprise Linux System Role in einem Playbook

### Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab role-review start` aus. Das Skript erstellt das Arbeitsverzeichnis `/home/student/role-review` und füllt es mit einer Ansible-Konfigurationsdatei, dem Hostinventar und anderen Lab-Dateien.

```
[student@workstation ~]$ lab role-review start
```

### Anweisungen

Ihr Unternehmen muss einen einzelnen Webserver bereitstellen, um Entwicklungscode für alle Webentwickler zu hosten. Sie sollen ein Playbook zur Konfiguration dieses Entwicklungs-Webservers schreiben.

Der Entwicklungs-Webserver muss mehrere Anforderungen erfüllen:

- Die Konfiguration des Entwicklungsservers stimmt mit der Konfiguration des Produktionsservers überein. Der Produktionsserver ist mit einer Ansible-Rolle konfiguriert, die vom Infrastruktur-Team des Unternehmens entwickelt wurde.
- Jeder Entwickler erhält auf dem Entwicklungsserver ein Verzeichnis, in dem er Code und Inhalt ablegt. Auf den Inhalt jedes Entwicklers wird über einen zugewiesenen, nicht dem Standard entsprechenden Port zugegriffen.
- SELinux ist auf „enforcing“ und „targeted“ festgelegt.

In Ihrem Playbook soll:

- Eine Rolle verwendet werden, um Verzeichnisse und Ports für jeden Entwickler auf dem Webserver zu konfigurieren. Sie müssen diese Rolle schreiben.

## Kapitel 7 | Playbooks mit Rollen vereinfachen

Diese Rolle hängt von einer Rolle ab, die vom Unternehmen zur Konfiguration von Apache entwickelt wurde. Sie sollen die Abhängigkeit anhand der Version v1.4 der Unternehmensrolle definieren. Die URL des Repository mit der Abhängigkeit lautet: `git@workstation.lab.example.com:infra/apache`

- Verwenden Sie die Rolle `rhel-system-roles.selinux` zum Konfigurieren von SELinux für die von Ihrem Webserver verwendeten nicht standardmäßigen HTTP-Ports. Sie erhalten eine `selinux.yml`-Variablendatei, die als `group_vars`-Datei installiert werden kann, um die korrekten Einstellungen an die Rolle zu übergeben.

1. Wechseln Sie in das Arbeitsverzeichnis `/home/student/role-review`.

```
[student@workstation ~]$ cd ~/role-review  
[student@workstation role-review]$
```

2. Erstellen Sie ein Playbook mit dem Namen `web_dev_server.yml` mit einem einzelnen Play namens `Configure Dev Web Server`. Konfigurieren Sie das Play so, dass es auf die Hostgruppe `dev_webserver` abzielt. Fügen Sie dem Play noch keine Rollen oder Aufgaben hinzu.

Stellen Sie sicher, dass das Play die Ausführung von Handlern erzwingt, da beim Entwickeln des Playbooks ein Fehler auftreten könnte.

Nach Abschluss enthält das Playbook `/home/student/role-review/web_dev_server.yml`:

```
---  
- name: Configure Dev Web Server  
  hosts: dev_webserver  
  force_handlers: yes
```

3. Prüfen Sie die Syntax des Playbooks. Führen Sie das Playbook aus. Die Syntaxprüfung sollte bestanden und das Playbook erfolgreich ausgeführt werden.

```
[student@workstation role-review]$ ansible-playbook \  
> --syntax-check web_dev_server.yml  
  
playbook: web_dev_server.yml  
[student@workstation role-review]$ ansible-playbook web_dev_server.yml  
PLAY [Configure Dev Web Server] *****  
  
TASK [Gathering Facts] *****  
ok: [servera.lab.example.com]  
  
PLAY RECAP *****  
servera.lab.example.com : ok=1    changed=0    unreachable=0    failed=0
```

4. Stellen Sie sicher, dass die Rollenabhängigkeiten des Playbooks installiert sind.

Die Rolle `apache.developer_configs`, die Sie erstellen sollen, hängt von der Rolle `infra.apache` ab. Erstellen Sie eine Datei `roles/requirements.yml`. Sie sollte die Rolle aus dem Git-Repository unter `git@workstation.lab.example.com:infra/apache` installieren, die Version v1.4 verwenden und lokal `infra.apache` benannt werden. Sie können davon ausgehen, dass Ihre SSH-Schlüssel so konfiguriert sind, dass Sie Rollen aus diesem Repository automatisch abrufen können. Installieren Sie die Rolle mit dem Befehl `ansible-galaxy`.

**Kapitel 7 |** Playbooks mit Rollen vereinfachen

Installieren Sie zusätzlich das Paket *rhel-system-roles*, falls noch nicht vorhanden.

- 4.1. Erstellen Sie für das Playbook-Projekt das Unterverzeichnis *roles*.

```
[student@workstation role-review]$ mkdir -v roles  
mkdir: created directory 'roles'
```

- 4.2. Erstellen Sie eine *roles/requirements.yml*-Datei, und fügen Sie einen Eintrag für die Rolle *infra.apache* hinzu. Verwenden Sie die Version v1.4 aus dem Git-Repository der Rolle.

Nach Abschluss enthält die Datei *roles/requirements.yml*:

```
- name: infra.apache  
  src: git@workstation.lab.example.com:infra/apache  
  scm: git  
  version: v1.4
```

- 4.3. Installieren Sie die Projektabhängigkeiten.

```
[student@workstation role-review]$ ansible-galaxy install \  
> -r roles/requirements.yml -p roles  
- extracting infra.apache to /home/student/role-review/roles/infra.apache  
- infra.apache (v1.4) was installed successfully
```

- 4.4. Installieren Sie das Paket „RHEL System Roles“, falls noch nicht vorhanden. Dies wurde während einer vorhergehenden Übung installiert.

```
[student@workstation role-review]$ sudo yum install rhel-system-roles
```

5. Initialisieren Sie im Unterverzeichnis *roles* eine neue Rolle mit dem Namen *apache.developer\_configs*.

Fügen Sie die Rolle *infra.apache* als Abhängigkeit für die neue Rolle hinzu, und verwenden Sie dafür dieselben Informationen für Name, Quelle, Version und Versionskontrollsystem wie die Datei *roles/requirements.yml*.

Die Datei *developer\_tasks.yml* im Projektverzeichnis enthält Aufgaben für die Rolle. Verschieben Sie diese Datei an den richtigen Speicherort, damit sie zur Aufgabendatei für diese Rolle wird, und ersetzen Sie die vorhandene Datei an diesem Speicherort.

Bei der Datei *developer.conf.j2* im Projektverzeichnis handelt es sich um eine Jinja2-Vorlage, die von der Aufgabendatei verwendet wird. Verschieben Sie sie an den richtigen Speicherort für die von dieser Rolle verwendeten Vorlagendateien.

- 5.1. Erstellen Sie mit *ansible-galaxy init* ein Rollengerüst für die Rolle *apache.developer\_configs*.

```
[student@workstation role-review]$ cd roles  
[student@workstation roles]$ ansible-galaxy init apache.developer_configs  
- apache.developer_configs was created successfully  
[student@workstation roles]$ cd ..  
[student@workstation role-review]$
```

- 5.2. Aktualisieren Sie die Datei `roles/apache.developer_configs/meta/main.yml` der Rolle `apache.developer_configs`, damit sie die Abhängigkeit von der Rolle `infra.apache` enthält.

Nach der Bearbeitung ist die Variable `dependencies` wie folgt definiert:

```
dependencies:
  - name: infra.apache
    src: git@workstation.lab.example.com:infra/apache
    scm: git
    version: v1.4
```

- 5.3. Überschreiben Sie die Datei `tasks/main.yml` der Rolle mit der Datei `developer_tasks.yml`.

```
[student@workstation role-review]$ mv -v developer_tasks.yml \
> roles/apache.developer_configs/tasks/main.yml
renamed 'developer_tasks.yml' -> 'roles/apache.developer_configs/tasks/main.yml'
```

- 5.4. Legen Sie die Datei `developer.conf.j2` im Verzeichnis `templates` der Rolle ab.

```
[student@workstation role-review]$ mv -v developer.conf.j2 \
> roles/apache.developer_configs/templates/
renamed 'developer.conf.j2' -> 'roles/apache.developer_configs/templates/
developer.conf.j2'
```

6. Die Rolle `apache.developer_configs` verarbeitet eine Liste von Benutzern, die in einer Variable mit dem Namen `web_developers` definiert sind. In der Datei `web_developers.yml` im Projektverzeichnis ist die Benutzerlistenvariable `web_developers` definiert. Überprüfen Sie diese Datei, und verwenden Sie sie zur Definition der Variable `web_developers` für die Hostgruppe des Entwicklungs-Webservers.

- 6.1. Überprüfen Sie die Datei `web_developers.yml`.

```
---
web_developers:
  - username: jdoe
    name: John Doe
    user_port: 9081
  - username: jdoe2
    name: Jane Doe
    user_port: 9082
```

Für jeden Webentwickler wird ein `name`, `username`, `user_port` definiert.

- 6.2. Legen Sie die Datei `web_developers.yml` im Unterverzeichnis `group_vars/dev_webserver` ab.

**Kapitel 7 |** Playbooks mit Rollen vereinfachen

```
[student@workstation role-review]$ mkdir -pv group_vars/dev_webserver
mkdir: created directory 'group_vars'
mkdir: created directory 'group_vars/dev_webserver'
[student@workstation role-review]$ mv -v web_developers.yml \
> group_vars/dev_webserver/
renamed 'web_developers.yml' -> 'group_vars/dev_webserver/web_developers.yml'
```

7. Fügen Sie dem Play im Playbook `web_dev_server.yml` die Rolle `apache.developer_configs` hinzu.

Das bearbeitete Playbook:

```
---
- name: Configure Dev Web Server
  hosts: dev_webserver
  force_handlers: yes
  roles:
    - apache.developer_configs
```

8. Prüfen Sie die Syntax des Playbooks. Führen Sie das Playbook aus. Die Syntaxprüfung sollte bestanden werden, das Playbook sollte jedoch fehlschlagen, wenn die Rolle `infra.apache` versucht, Apache HTTPD neu zu starten.

```
[student@workstation role-review]$ ansible-playbook \
> --syntax-check web_dev_server.yml

playbook: web_dev_server.yml
[student@workstation role-review]$ ansible-playbook web_dev_server.yml

PLAY [Configure Dev Web Server] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

...output omitted...

TASK [infra.apache : Install a skeleton index.html] ****
skipping: [servera.lab.example.com]

TASK [apache.developer_configs : Create user accounts] ****
changed: [servera.lab.example.com] => (item={u'username': u'jdoe', u'user_port': 9081, u'name': u'John Doe'})
changed: [servera.lab.example.com] => (item={u'username': u'jdoe2', u'user_port': 9082, u'name': u'Jane Doe'})

...output omitted...

RUNNING HANDLER [infra.apache : restart firewalld] ****
changed: [servera.lab.example.com]

RUNNING HANDLER [infra.apache : restart apache] ****
```

```

fatal: [servera.lab.example.com]: FAILED! => {"changed": false, "msg": "Unable to
restart service httpd: Job for httpd.service failed because the control process
exited with error code. See \\\"systemctl status httpd.service\\\" and \\\"journalctl -
xe\\\" for details.\n"}

NO MORE HOSTS LEFT *****
      to retry, use: --limit @/home/student/role-review/web_dev_server.retry

PLAY RECAP *****
servera.lab.example.com      : ok=13    changed=11    unreachable=0    failed=1
skipped=1      rescued=0     ignored=0

```

Es tritt ein Fehler auf, wenn der Service `httpd` neu gestartet wird. Der Service-Daemon `httpd` kann aufgrund des SELinux-Kontexts dieser Ports nicht an die nicht dem Standard entsprechenden HTTP-Ports gebunden werden.

- Apache HTTPD konnte im vorherigen Schritt nicht neu gestartet werden, da die für Ihre Entwickler verwendeten Netzwerkports mit den falschen SELinux-Kontexten gekennzeichnet sind. Sie haben die Variablendatei `selinux.yml` erhalten, die mit der Rolle `rhel-system-roles.selinux` verwendet werden kann, um das Problem zu beheben.

Erstellen Sie im Playbook `web_dev_server.yml` den Abschnitt `pre_tasks` für Ihr Play. Verwenden Sie in diesem Abschnitt eine Aufgabe, um die Rolle `rhel-system-roles.selinux` in eine `block/rescue`-Struktur einzubeziehen, damit sie richtig angewendet wird. Lesen Sie das Kursskript oder die Dokumentation für diese Rolle, um zu erfahren, wie Sie diese Aufgabe durchführen können.

Überprüfen Sie die Datei `selinux.yml`. Verschieben Sie sie an den richtigen Speicherort, damit die Variablen für die Hostgruppe `dev_webserver` festgelegt werden.

- Der Abschnitt `pre_tasks` kann dem Playbook `web_dev_server.yml` am Ende des Plays hinzugefügt werden.

Sie können sich den Block in `/usr/share/doc/rhel-system-roles/selinux/example-selinux-playbook.yml` ansehen, um eine grundlegende Übersicht über die Anwendung der Rolle zu erhalten. Ersetzen Sie die komplexe `shell`- und `wait_for_connection`-Logik durch das `reboot`-Modul.

Der Abschnitt `pre_tasks` sollte Folgendes enthalten:

```

pre_tasks:
  - name: Check SELinux configuration
    block:
      - include_role:
          name: rhel-system-roles.selinux
    rescue:
      # Fail if failed for a different reason than selinux_reboot_required.
      - name: Check for general failure
        fail:
          msg: "SELinux role failed."
        when: not selinux_reboot_required

      - name: Restart managed host
        reboot:
          msg: "Ansible rebooting system for updates."

```

**Kapitel 7 |** Playbooks mit Rollen vereinfachen

```
- name: Reapply SELinux role to complete changes
  include_role:
    name: rhel-system-roles.selinux
```

- 9.2. Die Datei `selinux.yml` enthält Variablendefinitionen für die Rolle `rhel-system-roles.selinux`. Verwenden Sie die Datei, um Variablen für die Hostgruppe des Plays zu definieren.

```
[student@workstation role-review]$ cat selinux.yml
---
# variables used by rhel-system-roles.selinux

selinux_policy: targeted
selinux_state: enforcing

selinux_ports:
  - ports:
      - "9081"
      - "9082"
    proto: 'tcp'
    setype: 'http_port_t'
    state: 'present'

[student@workstation role-review]$ mv -v selinux.yml \
> group_vars/dev_webserver/
renamed 'selinux.yml' -> 'group_vars/dev_webserver/selinux.yml'
```

10. Überprüfen Sie das finale Playbook `web_dev_server.yml`, und führen Sie eine Syntaxprüfung durch. Die Syntaxprüfung sollte bestanden werden.

Das finale Playbook `web_dev_server.yml` sollte wie folgt lauten:

```
---
- name: Configure Dev Web Server
  hosts: dev_webserver
  force_handlers: yes
  roles:
    - apache.developer_configs
  pre_tasks:
    - name: Check SELinux configuration
      block:
        - include_role:
            name: rhel-system-roles.selinux
  rescue:
    # Fail if failed for a different reason than selinux_reboot_required.
    - name: Check for general failure
      fail:
        msg: "SELinux role failed."
        when: not selinux_reboot_required

    - name: Restart managed host
      reboot:
        msg: "Ansible rebooting system for updates."
```

```
- name: Reapply SELinux role to complete changes
  include_role:
    name: rhel-system-roles.selinux
```



### Anmerkung

Ob `pre_tasks` sich am Ende des Plays oder an der „richtigen“ Position in Bezug auf die Ausführungsreihenfolge in der Playbook-Datei befindet, ist für `ansible-playbook` unerheblich. Die Aufgaben des Plays werden trotzdem in der richtigen Reihenfolge ausgeführt.

Überprüfen Sie, ob das Playbook `web_dev_server.yml` eine Syntaxprüfung besteht.

```
[student@workstation role-review]$ ansible-playbook \
> --syntax-check web_dev_server.yml

playbook: web_dev_server.yml
```

11. Führen Sie das Playbook aus. Es sollte korrekt ausgeführt werden.

```
[student@workstation role-review]$ ansible-playbook web_dev_server.yml

PLAY [Configure Dev Web Server] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [include_role : rhel-system-roles.selinux] ****
TASK [rhel-system-roles.selinux : Install SELinux python3 tools] ****
ok: [servera.lab.example.com]

...output omitted...

TASK [infra.apache : Apache Service is started] ****
changed: [servera.lab.example.com]

...output omitted...

TASK [apache.developer_configs : Copy Per-Developer Config files] ****
ok: [servera.lab.example.com] => (item={'username': 'jdoe', 'name': 'John Doe', 'user_port': 9081})
ok: [servera.lab.example.com] => (item={'username': 'jdoe2', 'name': 'Jane Doe', 'user_port': 9082})

PLAY RECAP ****
servera.lab.example.com    : ok=19    changed=3    unreachable=0    failed=0
skipped=14    rescued=0    ignored=0
```

## Kapitel 7 | Playbooks mit Rollen vereinfachen

12. Testen Sie die Konfiguration des Entwicklungs-Webservers. Stellen Sie sicher, dass auf alle Endpunkte zugegriffen werden kann und sie den Inhalt der einzelnen Entwickler bereitstellen.

```
[student@workstation role-review]$ curl servera
This is the production server on servera.lab.example.com
[student@workstation role-review]$ curl servera:9081
This is index.html for user: John Doe (jdoe)
[student@workstation role-review]$ curl servera:9082
This is index.html for user: Jane Doe (jdoe2)
[student@workstation role-review]$
```

## Bewertung

Bewerten Sie Ihre Arbeit, indem Sie auf Ihrem workstation-Rechner den Befehl `lab role-review grade` ausführen. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie das Skript so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab role-review grade
```

## Beenden

Führen Sie auf workstation das Skript `lab role-review finish` aus, um diese Übung zu bereinigen.

```
[student@workstation ~]$ lab role-review finish
```

Hiermit ist die praktische Übung abgeschlossen.

# Zusammenfassung

---

In diesem Kapitel wurden die folgenden Themen behandelt:

- Rollen organisieren Ansible-Code derart, dass sie Wiederverwendung und Teilung ermöglichen.
- Red Hat Enterprise Linux System Roles sind eine Sammlung von getesteten und unterstützten Rollen, die Ihnen beim Konfigurieren von Hostsubsystemen für verschiedene Versionen von Red Hat Enterprise Linux helfen sollen.
- Ansible Galaxy [<https://galaxy.ansible.com>] ist eine öffentliche Library mit Ansible-Rollen, die von Ansible-Benutzern geschrieben wurden. Der Befehl `ansible-galaxy` kann weitere Informationen zu Rollen suchen, diese anzeigen, sowie Rollen installieren, auflisten, löschen oder initialisieren.
- Für ein Playbook erforderliche externe Rollen können in der Datei `roles/requirements.yml` definiert sein. Der Befehl `ansible-galaxy install -r roles/requirements.yml` verwendet diese Datei, um die Rollen auf dem Kontrollknoten zu installieren.



## Kapitel 8

# Fehlerbehebung in Ansible

### Ziel

Fehler in Playbooks und auf verwalteten Hosts beheben.

### Ziele

- Beheben Sie allgemeine Probleme mit einem neuen Playbook und reparieren Sie diese.
- Beheben Sie Fehler auf verwalteten Hosts, wenn Sie ein Playbook ausführen.

### Abschnitte

- Fehlerbehebung in Playbooks (und angeleitete Übung)
- Fehlerbehebung auf verwalteten Ansible-Hosts (und angeleitete Übung)

### Praktische Übung

- Fehlerbehebung in Ansible

# Fehlerbehebung in Playbooks

---

## Ziele

Am Ende dieses Abschnitts sollten Sie in der Lage sein, allgemeine Probleme mit neuen Playbooks zu finden und zu beheben.

## Protokolldateien für Ansible

Standardmäßig protokolliert Ansible seine Ausgaben nicht in Protokolldateien. Ansible stellt eine integrierte Protokoll-Infrastruktur bereit, die durch den Parameter `log_path` im Abschnitt `default` der Konfigurationsdatei `ansible.cfg` oder durch die Umgebungsvariable `$ANSIBLE_LOG_PATH` konfiguriert werden kann. Wenn eine davon oder beide konfiguriert sind, speichert Ansible Ausgaben der beiden Befehle `ansible` und `ansible-playbook` in der Protokolldatei, die entweder durch die Konfigurationsdatei `ansible.cfg` oder die Umgebungsvariable `$ANSIBLE_LOG_PATH` konfiguriert wurde.

Wenn Sie Ansible so konfigurieren, dass Protokolldateien in `/var/log` geschrieben werden, dann empfiehlt Red Hat, dass Sie `logrotate` für die Verwaltung der Ansible-Protokolldateien konfigurieren.

## Das Debug-Modul

Das `debug`-Modul bietet Einblick in das, was im Play passiert. Dieses Modul kann den Wert für eine bestimmte Variable an einem bestimmten Punkt im Play anzeigen. Diese Funktion ist wichtig für das Debuggen von Aufgaben, die über Variablen miteinander kommunizieren (z. B. die Verwendung der Ausgabe einer Aufgabe als Eingabe für die nachfolgende Aufgabe).

Die folgenden Beispiele verwenden die Einstellungen `msg` und `var` innerhalb von `debug`-Aufgaben. Das erste Beispiel zeigt den Wert zur Laufzeit des Fakts `ansible_facts['memfree_mb']` als Teil einer Nachricht an, die an die Ausgabe von `ansible-playbook` ausgegeben wird. Das zweite Beispiel zeigt den Wert der Variablen `output`.

```
- name: Display free memory
  debug:
    msg: "Free memory for this system is {{ ansible_facts['memfree_mb'] }}"

- name: Display the "output" variable
  debug:
    var: output
    verbosity: 2
```

## Fehlermanagement

Es gibt mehrere Probleme, die während der Ausführung eines Playbooks auftreten können, hauptsächlich aufgrund der Syntax des Playbooks oder der Syntax einer verwendeten Vorlage oder aufgrund von Problemen bei der Verbindung mit den verwalteten Hosts (z. B. ein Fehler im Hostnamen des verwalteten Host in der Inventardatei). Diese Fehler werden zum Zeitpunkt der Ausführung mit dem Befehl `ansible-playbook` ausgegeben.

## Kapitel 8 | Fehlerbehebung in Ansible

Sie haben zuvor in diesem Kurs etwas über die Option `--syntax-check` erfahren, die die YAML-Syntax für das Playbook prüft. Es empfiehlt sich, eine Syntaxprüfung für Ihr Playbook durchzuführen, bevor Sie es verwenden oder wenn Probleme mit dem Playbook auftreten.

```
[student@demo ~]$ ansible-playbook play.yml --syntax-check
```

Sie können auch die Option `--step` verwenden, um ein Playbook aufgabenweise zu durchlaufen. Der Befehl `ansible-playbook --step` fordert Sie interaktiv zur Bestätigung auf, dass jede Aufgabe ausgeführt werden soll.

```
[student@demo ~]$ ansible-playbook play.yml --step
```

Die Option `--start-at-task` gestattet es Ihnen, die Ausführung eines Playbooks über eine bestimmte Aufgabe zu starten. Als Argument wird der Name der Aufgabe übernommen, bei der begonnen werden soll.

```
[student@demo ~]$ ansible-playbook play.yml --start-at-task="start httpd service"
```

## Debuggen

Die Ausgabe eines Playbooks, das mit dem Befehl `ansible-playbook` ausgeführt wurde, ist ein guter Ausgangspunkt zum Beheben von Fehlern, die von Ansible verwaltete Hosts betreffen. Betrachten Sie die folgende Ausgabe einer Playbook-Ausführung:

```
PLAY [Service Deployment] *****
...output omitted...
TASK: [Install a service] *****
ok: [demoservera]
ok: [demoserverb]

PLAY RECAP *****
demoservera      : ok=2    changed=0    unreachable=0   failed=0
demoserverb      : ok=2    changed=0    unreachable=0   failed=0
```

Die obige Ausgabe enthält einen PLAY-Header mit dem Namen des auszuführenden Plays gefolgt von einem oder mehreren TASK-Headern. Jeder dieser Header steht für die zugehörige *Aufgabe* im Playbook. Diese Aufgabe wird auf allen verwalteten Hosts ausgeführt, die zu der im Parameter `hosts` des Playbooks angegebenen Gruppe gehören.

Da alle verwalteten Hosts die Aufgaben der einzelnen Plays ausführen, wird der Name des verwalteten Hosts unter dem entsprechenden TASK-Header zusammen mit dem Status der Aufgabe auf dem verwalteten Host angezeigt. Als Status der Aufgabe kann `ok`, `fatal`, `changed` oder `skipping` angezeigt werden.

Am Ende der Ausgabe für die einzelnen Plays zeigt der Abschnitt `PLAY RECAP` die Anzahl der für jeden verwalteten Host ausgeführten Aufgaben an.

Wie bereits zuvor in diesem Kurs erläutert, können Sie die Ausführlichkeit der Ausgabe von `ansible-playbook` erhöhen, indem Sie eine oder mehrere `-v`-Optionen hinzufügen. Mit dem Befehl `ansible-playbook -v` werden zusätzliche Debugging-Informationen mit bis zu vier Gesamtebenen ausgegeben.

## Konfiguration von ausführlichen Informationen

Option	Beschreibung
-v	Die Ausgabedaten werden angezeigt.
-vv	Die Ausgabe- und die Eingabedaten werden angezeigt.
-vvv	Schließt Information über die Verbindungen zu verwalteten Hosts ein.
-vvvv	Schließt zusätzliche Informationen wie die auf den einzelnen Remote-Hosts ausgeführten Skripts sowie den Benutzer ein, der die einzelnen Skripts ausführt.

## Empfohlene Verfahren für die Verwaltung von Playbooks

Obwohl die zuvor erörterten Tools Sie beim Ermitteln und Beheben von Problemen in Playbooks unterstützen können, ist es beim Entwickeln dieser Playbooks dennoch wichtig, sich an empfohlene Verfahren zu halten, durch die die Fehlerbehebung vereinfacht werden kann. Im Folgenden sind einige bewährte Verfahren für die Entwicklung von Playbooks aufgeführt:

- Verwenden Sie eine kurze Beschreibung zum Zweck des Plays oder der Aufgabe, um Plays und Aufgaben zu benennen. Der Name des Plays oder der Aufgabe wird bei der Ausführung des Playbooks angezeigt. Dies hilft auch bei der Dokumentierung, was mit den einzelnen Plays oder Aufgaben erreicht werden soll und warum dies möglicherweise notwendig ist.
- Fügen Sie Kommentare mit zusätzlichen Informationen über die Aufgaben ein.
- Verwenden Sie vertikale Leerzeichen effektiv. Ordnen Sie Aufgabenattribute im Allgemeinen vertikal an, damit sie einfacher gelesen werden können.
- Die konsequente horizontale Einrückung ist sehr wichtig. Verwenden Sie Leerzeichen und keine Tabulatoren, um Einrückungsfehler zu vermeiden. Richten Sie Ihren Texteditor so ein, dass Leerzeichen eingefügt werden, wenn Sie die TAB-Taste drücken, um dies zu vereinfachen.
- Versuchen Sie, das Playbook so einfach wie möglich zu gestalten. Verwenden Sie nur die Funktionen, die Sie benötigen.



### Literaturhinweise

#### Configuring Ansible – Ansible-Dokumentation

[https://docs.ansible.com/ansible/2.9/installation\\_guide/intro\\_configuration.html](https://docs.ansible.com/ansible/2.9/installation_guide/intro_configuration.html)

#### debug – Print statements during execution – Ansible-Dokumentation

[https://docs.ansible.com/ansible/2.9/modules/debug\\_module.html](https://docs.ansible.com/ansible/2.9/modules/debug_module.html)

#### Best Practices – Ansible-Dokumentation

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_best\\_practices.html](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_best_practices.html)

## ► Angeleitete Übung

# Fehlerbehebung in Playbooks

In dieser Übung führen Sie die Fehlerbehebung für ein an Sie übergebenes Playbook durch, das nicht ordnungsgemäß funktioniert.

## Ergebnisse

Sie sollten in der Lage sein, Probleme in Playbooks zu beheben.

## Bevor Sie Beginnen

Melden Sie sich als student mit dem Passwort student bei workstation an.

Führen Sie auf workstation das Skript lab troubleshoot-playbook start aus. Es stellt sicher, dass Ansible auf workstation installiert ist. Es erstellt außerdem das Verzeichnis /home/student/troubleshoot-playbook/ und lädt in dieses Verzeichnis die Dateien inventory, samba.yml und samba.conf.j2 von http://materials.example.com/labs/troubleshoot-playbook/ herunter.

```
[student@workstation ~]$ lab troubleshoot-playbook start
```

## Anweisungen

- 1. Wechseln Sie auf workstation zum Verzeichnis /home/student/troubleshoot-playbook/.

```
[student@workstation ~]$ cd ~/troubleshoot-playbook/  
[student@workstation troubleshoot-playbook]$
```

- 2. Erstellen Sie im aktuellen Verzeichnis eine Datei mit dem Namen ansible.cfg. Dadurch sollte der Parameter log\_path so festgelegt werden, dass Ansible-Protokolle in die Datei /home/student/troubleshoot-playbook/ansible.log geschrieben werden. Der Parameter inventory sollte so festgelegt werden, dass die Datei /home/student/troubleshoot-playbook/inventory verwendet wird, die durch das Übungsskript bereitgestellt wurde.

Wenn Sie fertig sind, sollte ansible.cfg Folgendes enthalten:

```
[defaults]  
log_path = /home/student/troubleshoot-playbook/ansible.log  
inventory = /home/student/troubleshoot-playbook/inventory
```

- 3. Führen Sie das Playbook aus. Es wird ein Fehler auftreten.

Dieses Playbook würde einen Samba-Server einrichten, wenn alles korrekt ausgeführt wurde. Die Ausführung schlägt jedoch fehl, weil die doppelten Anführungszeichen für die Variablendefinition random\_var fehlen. Lesen Sie die Fehlermeldung, um zu erfahren, wie ansible-playbook das Problem meldet. Der Variable random\_var wird ein Wert zugewiesen, der einen Doppelpunkt enthält und nicht in Anführungszeichen gesetzt ist.

**Kapitel 8 |** Fehlerbehebung in Ansible

```
[student@workstation troubleshoot-playbook]$ ansible-playbook samba.yml
ERROR! We were unable to read either as JSON nor YAML, these are the errors we got
from each:
JSON: Expecting value: line 1 column 1 (char 0)

Syntax Error while loading YAML.
  mapping values are not allowed in this context

The error appears to be in '/home/student/troubleshoot-playbook/samba.yml': line
8, column 30, but may be elsewhere in the file depending on the exact syntax
problem.

The offending line appears to be:

  install_state: installed
    random_var: This is colon: test
                  ^ here
```

- 4. Überprüfen Sie, ob der Fehler ordnungsgemäß in der Datei /home/student/troubleshoot-playbook/ansible.log protokolliert wurde.

```
[student@workstation troubleshoot-playbook]$ tail ansible.log

The error appears to be in '/home/student/troubleshoot-playbook/samba.yml': line
8, column 30, but may be elsewhere in the file depending on the exact syntax
problem.

The offending line appears to be:

  install_state: installed
    random_var: This is colon: test
                  ^ here
```

- 5. Korrigieren Sie den Fehler im Playbook samba.yml, indem Sie den gesamten der Variable random\_var zugewiesenen Wert in Anführungszeichen setzen. Die korrigierte Version des Playbooks enthält den folgenden Inhalt:

```
...output omitted...
vars:
  install_state: installed
  random_var: "This is colon: test"
...output omitted...
```

- 6. Prüfen Sie das Playbook mit der Option `--syntax-check`. Es wird ein weiterer Fehler aufgrund eines zusätzlichen Leerzeichens bei der Einrückung für die letzte Aufgabe `deliver samba config` ausgegeben.

```
[student@workstation troubleshoot-playbook]$ ansible-playbook --syntax-check \
> samba.yml
ERROR! We were unable to read either as JSON nor YAML, these are the errors we got
from each:
JSON: Expecting value: line 1 column 1 (char 0)

Syntax Error while loading YAML.
  did not find expected key

The error appears to be in '/home/student/troubleshoot-playbook/samba.yml': line
44, column 4, but may be elsewhere in the file depending on the exact syntax
problem.

The offending line appears to be:

  - name: deliver samba config
    ^ here
```

- 7. Entfernen Sie im Playbook das zusätzliche Leerzeichen aus allen Zeilen dieser Aufgabe. Das korrigierte Playbook sollte wie folgt angezeigt werden:

```
...output omitted...
- name: configure firewall for samba
  firewalld:
    state: enabled
    permanent: true
    immediate: true
    service: samba

- name: deliver samba config
  template:
    src: templates/samba.conf.j2
    dest: /etc/samba/smb.conf
    owner: root
    group: root
    mode: 0644
```

- 8. Führen Sie das Playbook mit der Option `--syntax-check` aus. Es wird ein Fehler ausgegeben, weil die Variable `install_state` als Parameter in der Aufgabe `install samba` verwendet wurde. Sie ist nicht in Anführungszeichen gesetzt.

```
[student@workstation troubleshoot-playbook]$ ansible-playbook --syntax-check \
> samba.yml
ERROR! We were unable to read either as JSON nor YAML, these are the errors we got
from each:
JSON: Expecting value: line 1 column 1 (char 0)

Syntax Error while loading YAML.
```

**Kapitel 8 |** Fehlerbehebung in Ansible

```
found unacceptable key (unhashable type: 'AnsibleMapping')

The error appears to be in '/home/student/troubleshoot-playbook/samba.yml': line
14, column 15, but may be elsewhere in the file depending on the exact syntax
problem.
```

The offending line appears to be:

```
name: samba
state: {{ install_state }}
      ^ here
```

We could be wrong, but this one looks like it might be an issue with missing quotes. Always quote template expression brackets when they start a value. For instance:

```
with_items:
- {{ foo }}
```

Should be written as:

```
with_items:
- "{{ foo }}"
```

- 9. Korrigieren Sie im Playbook die Aufgabe `install samba`. Der Verweis auf die Variable `install_state` muss in Anführungszeichen gesetzt werden. Der Inhalt der Datei sollte wie folgt aussehen:

```
...output omitted...
tasks:
- name: install samba
  yum:
    name: samba
    state: "{{ install_state }}"
...output omitted...
```

- 10. Führen Sie das Playbook mit der Option `--syntax-check` aus. Jetzt sollten keine weiteren Syntaxfehler angezeigt werden.

```
[student@workstation troubleshoot-playbook]$ ansible-playbook --syntax-check \
> samba.yml

playbook: samba.yml
```

- 11. Führen Sie das Playbook aus. Ein Fehler im Zusammenhang mit SSH wird ausgegeben.

```
[student@workstation troubleshoot-playbook]$ ansible-playbook samba.yml
PLAY [Install a samba server] ****
TASK [Gathering Facts] ****
fatal: [servera.lab.exammples.com]: UNREACHABLE! => {"changed": false,
  "msg": "Failed to connect to the host via ssh: ssh: connect to host
servera.lab.exammples.com port 22: Connection timed out", "unreachable": true}

PLAY RECAP ****
servera.lab.exammples.com    : ok=0      changed=0      unreachable=1      failed=0  ...
```

- 12. Prüfen Sie mit dem Befehl ping, ob der verwaltete Host `servera.lab.example.com` ausgeführt wird.

```
[student@workstation troubleshoot-playbook]$ ping -c3 servera.lab.example.com
PING servera.lab.example.com (172.25.250.10) 56(84) bytes of data.
64 bytes from servera.lab.example.com (172.25.250.10): icmp_seq=1 ttl=64
time=0.247 ms
64 bytes from servera.lab.example.com (172.25.250.10): icmp_seq=2 ttl=64
time=0.329 ms
64 bytes from servera.lab.example.com (172.25.250.10): icmp_seq=3 ttl=64
time=0.320 ms

--- servera.lab.example.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.247/0.298/0.329/0.041 ms
```

- 13. Überprüfen Sie, ob Sie sich als devops-Benutzer mit SSH bei dem verwalteten Host `servera.lab.example.com` anmelden können, und dass die richtigen SSH-Keywords vorhanden sind. Melden Sie sich wieder ab, wenn Sie fertig sind.

```
[student@workstation troubleshoot-playbook]$ ssh devops@servera.lab.example.com
Activate the web console with: systemctl enable --now cockpit.socket
...output omitted...
[devops@servera ~]$ exit
logout
Connection to servera.lab.example.com closed.
```

- 14. Führen Sie das Playbook mit `-vvvv` erneut aus, um weitere Informationen über die Ausführung zu erhalten. Es wird ein Fehler ausgegeben, weil der verwaltete Host `servera.lab.example.com` nicht erreichbar ist.

```
[student@workstation troubleshoot-playbook]$ ansible-playbook -vvvv samba.yml
...output omitted...

PLAYBOOK: samba.yml ****
1 plays in samba.yml

PLAY [Install a samba server] ****
```

```
TASK [Gathering Facts] *****
task path: /home/student/troubleshoot-playbook/samba.yml:2
<servera.lab.exammples.com> ESTABLISH SSH CONNECTION FOR USER: devops
...output omitted...
fatal: [servera.lab.exammples.com]: UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: OpenSSH_8.0p1, OpenSSL ...
Control socket \"/home/student/.ansible/cp/d4775f48c9\" does not exist\r\ndebug2:
resolving \"servera.lab.exammples.com\" port 22\r\ndebug2: ssh_connect_direct
\r\ndebug1: Connecting to servera.lab.exammples.com [3.223.115.185] port 22.\r
\ndebug2: fd 4 setting O_NONBLOCK\r\ndebug1: connect to address 3.223.115.185 port
22: Connection timed out\r\nnssh: connect to host servera.lab.exammples.com port
22: Connection timed out",
    "unreachable": true
}

...output omitted...
PLAY RECAP *****
servera.lab.exammples.com : ok=0      changed=0      unreachable=1      failed=0
```

- 15. Wenn Sie mit Ansible die höchste Ausführlichkeitsebene verwenden, ist die Überprüfung der Ansible-Protokolldatei besser geeignet als die Überprüfung der Konsolenausgabe. Sie können die Protokolldatei mit dem Befehl less anzeigen oder mit dem Befehl grep nach Mustern in der Protokolldatei suchen. Suchen Sie in der Datei/home/student/troubleshoot-playbook/ansible.log nach dem Wort fatal.

```
[student@workstation troubleshoot-playbook]$ grep -i fatal ansible.log
2021-07-15 13:56:21,766 p=45752 u=student n=ansible | fatal:
[servera.lab.exammpble.com]: UNREACHABLE! => {"changed": false, "msg": "Failed to
connect to the host via ssh: ssh: connect to host servera.lab.exammpble.com port
22: Connection timed out", "unreachable": true}
2021-07-15 14:22:43,262 p=46055 u=student n=ansible | fatal:
[servera.lab.exammpble.com]: UNREACHABLE! => {
```

- 16. Untersuchen Sie die Datei `inventory` auf Fehler. Beachten Sie, dass in der Gruppe [samba\_servers] der Host `servera.lab.example.com` falsch geschrieben ist. Korrigieren Sie diesen Fehler wie folgt:

```
[samba_servers]  
servera.lab.example.com  
...output omitted...
```

- 17. Führen Sie das Playbook erneut aus. Die Aufgabe `debug install_state variable` gibt die Meldung *The state for the samba service is installed* zurück. Diese Aufgabe verwendet das Modul `debug` und zeigt den Wert der Variable `install_state` an. Es wird auch ein Fehler in der Aufgabe `deliver samba config` angezeigt, da die Datei `samba.j2` nicht im Arbeitsverzeichnis `/home/student/troubleshoot-playbook/` verfügbar ist.

```
[student@workstation troubleshoot-playbook]$ ansible-playbook samba.yml
```

```
PLAY [Install a samba server] ****
...output omitted...
```

```

TASK [debug install_state variable] ****
ok: [servera.lab.example.com] => {
    "msg": "The state for the samba service is installed"
}
...output omitted...
TASK [deliver samba config] ****
fatal: [servera.lab.example.com]: FAILED! => {"changed": false, "msg": "Could not
  find or access 'samba.j2'\nSearched in:\n\t/home/student/troubleshoot-playbook/
  templates/samba.j2\n\t/home/student/troubleshoot-playbook/samba.j2\n\t/home/
  student/troubleshoot-playbook/templates/samba.j2\n\t/home/student/troubleshoot-
  playbook/samba.j2 on the Ansible Controller.\nIf you are using a module and expect
  the file to exist on the remote, see the remote_src option"}
...output omitted...
PLAY RECAP ****
servera.lab.example.com : ok=7    changed=3    unreachable=0    failed=1 ...

```

- 18. Legen Sie im Playbook für den Parameter `src` in der Aufgabe `deliver samba config` den Wert `samba.conf.j2` fest. Wenn Sie fertig sind, sollte das Playbook wie folgt aussehen:

```

...output omitted...
- name: deliver samba config
  template:
    src: samba.conf.j2
    dest: /etc/samba/smb.conf
    owner: root
...output omitted...

```

- 19. Führen Sie das Playbook erneut aus. Führen Sie das Playbook mit der Option `--step` aus. Es sollte fehlerfrei ausgeführt werden.

```

[student@workstation troubleshoot-playbook]$ ansible-playbook samba.yml --step

PLAY [Install a samba server] ****
Perform task: TASK: Gathering Facts (N)o/(y)es/(c)ontinue: y
...output omitted...
Perform task: TASK: install samba (N)o/(y)es/(c)ontinue: y
...output omitted...
Perform task: TASK: install firewalld (N)o/(y)es/(c)ontinue: y
...output omitted...
Perform task: TASK: debug install_state variable (N)o/(y)es/(c)ontinue: y
...output omitted...
Perform task: TASK: start samba (N)o/(y)es/(c)ontinue: y
...output omitted...
Perform task: TASK: start firewalld (N)o/(y)es/(c)ontinue: y
...output omitted...
Perform task: TASK: configure firewall for samba (N)o/(y)es/(c)ontinue: y
...output omitted...
Perform task: TASK: deliver samba config (N)o/(y)es/(c)ontinue: y
...output omitted...
PLAY RECAP ****
servera.lab.example.com : ok=8    changed=1    unreachable=0    failed=0

```

## Beenden

Führen Sie auf `workstation` das Skript `lab troubleshoot-playbook finish` aus, um diese Übung zu bereinigen.

```
[student@workstation ~]$ lab troubleshoot-playbook finish
```

Hiermit ist die angeleitete Übung beendet.

# Fehlerbehebung auf verwalteten Ansible-Hosts

## Ziele

Am Ende dieses Abschnitts sollten Sie zu Folgendem in der Lage sein: Fehler beim Ausführen eines Playbooks auf verwalteten Hosts behandeln.

## Verwenden des Check-Modus als Testtool

Mit dem Befehl `ansible-playbook --check` können Sie für ein Playbook „Feuerproben“ (Smoke Tests) durchführen. Diese Option führt das Playbook aus, ohne Änderungen an der Konfiguration der verwalteten Hosts vorzunehmen. Wenn ein im Playbook verwendetes Modul den *Check-Modus* unterstützt, werden die Änderungen, die an den verwalteten Hosts vorgenommen würden, angezeigt, aber nicht ausgeführt. Wenn der Check-Modus jedoch nicht von einem Modul unterstützt wird, werden die Änderungen nicht angezeigt und das Modul führt dennoch keine Aktion aus.

```
[student@demo ~]$ ansible-playbook --check playbook.yml
```



### Anmerkung

Der Befehl `ansible-playbook --check` wird möglicherweise nicht ordnungsgemäß ausgeführt, wenn in Ihren Aufgaben Bedingungen verwendet werden.

Sie können mit der Einstellung `check_mode` auch steuern, ob einzelne Aufgaben im Check-Modus ausgeführt werden. Wenn für eine Aufgabe `check_mode: yes` festgelegt ist, wird sie immer im Check-Modus ausgeführt, unabhängig davon, ob die Option `--check` an `ansible-playbook` übergeben wurde. Wenn für eine Aufgabe `check_mode: no` festgelegt ist, wird sie immer normal ausgeführt, auch wenn `--check` an `ansible-playbook` übergeben wird.

Die folgende Aufgabe wird immer im Check-Modus ausgeführt und sie nimmt keine Änderungen vor.

```
tasks:
  - name: task always in check mode
    shell: uname -a
    check_mode: yes
```

Die folgende Aufgabe wird immer normal ausgeführt, auch wenn sie mit `ansible-playbook --check` gestartet wird.

```
tasks:
  - name: task always runs even in check mode
    shell: uname -a
    check_mode: no
```

Dies kann hilfreich sein, da Sie den Großteil eines Playbooks normal ausführen können, während Sie einzelne Aufgaben mit `check_mode: yes` testen. Ebenso können Sie dafür sorgen, dass Testläufe im Check-Modus eher angemessene Ergebnisse liefern, indem Sie ausgewählte Aufgaben ausführen, die Fakten sammeln oder Variablen für Bedingungen festlegen, ohne dass die verwalteten Hosts mit `check_mode: no` geändert werden.

Eine Aufgabe kann ermitteln, ob das Playbook im Check-Modus ausgeführt wird, indem der Wert der magischen Variablen `ansible_check_mode` getestet wird. Diese boolesche Variable wird auf `true` festgelegt, wenn das Playbook im Check-Modus ausgeführt wird.



### Warnung

Aufgaben, für die `check_mode: no` festgelegt ist, werden auch ausgeführt, wenn das Playbook mit `ansible-playbook --check` ausgeführt wird. Deshalb können Sie nicht darauf vertrauen, dass die Option `--check` keine Änderungen an den verwalteten Hosts vornimmt, ohne dies durch Überprüfen des Playbooks und der damit verbundenen Rollen oder Aufgaben zu bestätigen.



### Anmerkung

Wenn Sie über ältere Playbooks verfügen, die `always_run: yes` verwenden, um die normale Ausführung von Aufgaben auch im Check-Modus zu erzwingen, müssen Sie diesen Code in Ansible 2.6 und höher durch `check_mode: no` ersetzen.

Der Befehl `ansible-playbook` bietet auch eine `--diff`-Option. Diese Option meldet die Änderungen, die an Vorlagendateien auf verwalteten Host vorgenommen wurden. Bei Verwendung der Option `--check` werden diese Änderungen in der Ausgabe des Befehls angezeigt, aber nicht tatsächlich ausgeführt.

```
[student@demo ~]$ ansible-playbook --check --diff playbook.yml
```

## Testen mit Modulen

Bestimmte Module liefern zusätzliche Informationen über den Status eines verwalteten Hosts. Die folgende Liste enthält einige Ansible-Module, die zum Testen und zur Fehlerbehebung auf verwalteten Hosts eingesetzt werden können.

- Das Modul `uri` ermöglicht die Überprüfung, ob eine RESTful API den erforderlichen Inhalt zurückgibt.

```
tasks:  
  - uri:  
    url: http://api.myapp.com  
    return_content: yes  
    register: apiresponse  
  
  - fail:  
    msg: 'version was not provided'  
    when: "'version' not in apiresponse.content"
```

- Das Modul `script` unterstützt die Ausführung eines Skripts auf einem verwalteten Host und schlägt fehl, wenn der Rückgabecode für dieses Skript ungleich null ist. Das Skript muss sich

## Kapitel 8 | Fehlerbehebung in Ansible

auf dem Kontrollknoten befinden und wird auf den verwalteten Host übertragen und dort ausgeführt.

```
tasks:  
  - script: check_free_memory
```

- Das Modul **stat** sammelt ähnlich wie der Befehl **stat** Fakten für eine Datei. Sie können damit eine Variable registrieren und anschließend testen, ob die Datei vorhanden ist, oder andere Informationen zu der Datei abrufen. Wenn die Datei nicht vorhanden ist, schlägt die Aufgabe **stat** nicht fehl, aber die registrierte Variable meldet **false** für `*.stat.exists`.

In diesem Beispiel wird eine Anwendung weiterhin ausgeführt, wenn `/var/run/app.lock` vorhanden ist. In diesem Fall sollte das Play abgebrochen werden.

```
tasks:  
  - name: Check if /var/run/app.lock exists  
    stat:  
      path: /var/run/app.lock  
    register: lock  
  
  - name: Fail if the application is running  
    fail:  
      when: lock.stat.exists
```

- Das Modul **assert** ist eine Alternative für das Modul **fail**. Das Modul **assert** unterstützt eine **that**-Option, die eine Liste von Bedingungen übernimmt. Wenn eine dieser Bedingungen nicht zutrifft, schlägt die Aufgabe fehl. Sie können die ausgegebene Nachricht mithilfe der Optionen **success\_msg** und **fail\_msg** anpassen, wenn ein Erfolg oder Fehler gemeldet wird.

Das folgende Beispiel ähnelt dem vorherigen, verwendet aber **assert** anstelle von **fail**.

```
tasks:  
  - name: Check if /var/run/app.lock exists  
    stat:  
      path: /var/run/app.lock  
    register: lock  
  
  - name: Fail if the application is running  
    assert:  
      that:  
        - not lock.stat.exists
```

## Fehlerbehebung bei Verbindungen

Viele allgemeine Probleme bei der Verwendung von Ansible zum Verwalten von Hosts sind von Verbindungen mit dem Host und Konfigurationsproblemen im Zusammenhang mit dem Remote-Benutzer und der Rechteerweiterung abhängig.

Wenn Probleme bei der Authentifizierung bei einem verwalteten Host auftreten, vergewissern Sie sich, dass `remote_user` in Ihrer Konfigurationsdatei oder im Play ordnungsgemäß festgelegt ist. Sie sollten zudem bestätigen, dass die richtigen SSH-Schlüssel eingerichtet sind oder Sie das richtige Passwort für diesen Benutzer bereitstellen.

## Kapitel 8 | Fehlerbehebung in Ansible

Stellen Sie sicher, dass `become` ordnungsgemäß festgelegt ist und Sie den richtigen `become_user` verwenden (dies ist standardmäßig `root`). Sie sollten bestätigen, dass Sie das richtige `sudo`-Passwort eingeben und dass `sudo` auf dem verwalteten Host ordnungsgemäß konfiguriert ist.

Ein weniger schwerwiegendes Problem betrifft die Inventareinstellungen. Bei einem komplexen Server mit mehreren Netzwerkadressen müssen Sie möglicherweise eine bestimmte Adresse oder einen DNS-Namen verwenden, wenn Sie eine Verbindung mit diesem System herstellen. Möglicherweise möchten Sie diese Adresse nicht als Inventarnamen des Rechners verwenden, um die Lesbarkeit zu verbessern. Sie können die Inventarvariable `ansible_host` für den Host festlegen, die den Inventarnamen mit einem anderen Namen oder einer anderen IP-Adresse überschreibt und von Ansible verwendet wird, um die Verbindung mit diesem Host herzustellen. Diese Variable könnte in der `host_vars`-Datei oder im Verzeichnis für diesen Host bzw. in der Inventardatei selbst festgelegt werden.

Der folgende Inventareintrag konfiguriert Ansible z. B. so, dass eine Verbindung mit `192.0.2.4` hergestellt wird, wenn der Host `web4.phx.example.com` verarbeitet wird:

```
web4.phx.example.com ansible_host=192.0.2.4
```

Auf diese Weise können Sie steuern, wie Ansible Verbindungen mit verwalteten Hosts herstellt. Es kann jedoch auch zu Problemen führen, wenn der Wert von `ansible_host` falsch ist.

## Testen verwalteter Hosts mithilfe von Ad-hoc-Befehlen

Die folgenden Beispiele zeigen einige der Prüfungen, die mit Ad-hoc-Befehlen für verwaltete Hosts erfolgen können.

Sie haben das Modul `ping` verwendet, um zu testen, ob Sie eine Verbindung mit verwalteten Hosts herstellen können. Abhängig von den von Ihnen übergebenen Optionen können Sie damit auch testen, ob Rechteerweiterung und Anmeldedaten ordnungsgemäß konfiguriert sind.

```
[student@demo ~]$ ansible demohost -m ping
demohost | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
[student@demo ~]$ ansible demohost -m ping --become
demohost | FAILED! => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "module_stderr": "sudo: a password is required\n",
    "module_stdout": "",
    "msg": "MODULE FAILURE\nSee stdout/stderr for the exact error",
    "rc": 1
}
```

Dieses Beispiel gibt den aktuell verfügbaren Speicherplatz auf den im verwalteten Host `demohost` konfigurierten Laufwerken zurück. Dies kann bei der Bestätigung hilfreich sein, dass das Dateisystem auf dem verwalteten Host nicht voll ist.

```
[student@demo ~]$ ansible demohost -m command -a 'df'
```

Dieses Beispiel gibt den aktuell verfügbaren Arbeitsspeicherplatz auf dem verwalteten Host `demohost` zurück.

```
[student@demo ~]$ ansible demohost -m command -a 'free -m'
```

## Die richtige Testebene

Ansible soll sicherstellen, dass die in Playbooks enthaltene und durch die Module vorgenommene Konfiguration richtig ist. Es überwacht alle Module hinsichtlich gemeldeter Fehler und stoppt das Playbook unverzüglich, sobald ein Fehler auftritt. Dies hilft dabei sicherzustellen, dass alle vor dem Auftreten des Fehlers ausgeführten Aufgaben fehlerfrei sind.

Aufgrund dessen besteht in der Regel kein Anlass zu überprüfen, ob das Ergebnis einer von Ansible verwalteten Aufgabe ordnungsgemäß auf die verwalteten Host angewendet wurde. Falls eine direktere Fehlerbehebung erforderlich ist, sollten Sie entweder Playbooks Health Checks hinzufügen oder diese direkt als Ad-hoc-Befehle ausführen. Sie sollten jedoch darauf achten, Ihre Aufgaben und Plays nicht zu komplex zu gestalten, um die von den Modulen selbst durchgeführten Tests doppelt zu überprüfen.



### Literaturhinweise

#### **Check Mode ("Dry Run") – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/user\\_guide/playbooks\\_checkmode.html](https://docs.ansible.com/ansible/2.9/user_guide/playbooks_checkmode.html)

#### **Testing Strategies – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/reference\\_appendices/test\\_strategies.html](https://docs.ansible.com/ansible/2.9/reference_appendices/test_strategies.html)

## ► Angeleitete Übung

# Fehlerbehebung auf verwalteten Ansible-Hosts

In dieser Übung behandeln Sie Aufgabenfehler, die auf einem Ihrer verwalteten Hosts auftreten, wenn Sie ein Playbook ausführen.

## Ergebnisse

Sie sollten die Fehlerbehebung für verwaltete Hosts durchführen können.

## Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` das Skript `lab troubleshoot-host start` aus. Es stellt zudem sicher, dass Ansible auf `workstation` installiert ist. Es lädt auch die Dateien `inventory`, `mailrelay.yml` und `postfix-relay-main.conf.j2` von `http://materials.example.com/labs/troubleshoot-host/` in das Verzeichnis `/home/student/troubleshoot-host/` herunter.

```
[student@workstation ~]$ lab troubleshoot-host start
```

## Anweisungen

- 1. Wechseln Sie auf `workstation` zum Verzeichnis `/home/student/troubleshoot-host/`.

```
[student@workstation ~]$ cd ~/troubleshoot-host/
[student@workstation troubleshoot-host]$
```

- 2. Führen Sie das Playbook `mailrelay.yml` im Check-Modus aus.

```
[student@workstation troubleshoot-host]$ ansible-playbook mailrelay.yml --check
PLAY [create mail relay servers] ****
...output omitted...
TASK [check main.cf file] ****
ok: [servera.lab.example.com]

TASK [verify main.cf file exists] ****
ok: [servera.lab.example.com]  => {
    "msg": "The main.cf file exists"
}
...output omitted...
TASK [email notification of always_bcc config] ****
fatal: [servera.lab.example.com]: FAILED! => {"msg": "The conditional check
'bcc_state.stdout != 'always_bcc =' failed. The error was: error while
evaluating conditional (bcc_state.stdout != 'always_bcc ='): 'dict object'
```

```

has no attribute 'stdout'\n\nThe error appears to have been in '/home/student/troubleshoot-host/mailrelay.yml': line 42, column 7, but may\nbe elsewhere in the file depending on the exact syntax problem.\n\nThe offending line appears to be:\n\n      - name: email notification of always_bcc config\n            ^ here\n"}\n...output omitted...
PLAY RECAP ****
servera.lab.example.com : ok=6    changed=3    unreachable=0    failed=1

```

Die Aufgabe *verify main.cf file exists* verwendet das Modul `stat`. Sie bestätigt, dass `main.cf` auf `servera.lab.example.com` vorhanden ist.

Die Aufgabe *email notification of always\_bcc config* ist fehlgeschlagen. Sie empfing von der Aufgabe *check for always\_bcc* keine Ausgabe, weil das Playbook im Check-Modus ausgeführt wurde.

- 3. Überprüfen Sie mit einem Ad-hoc-Befehl den Header der Datei `/etc/postfix/main.cf`.

```
[student@workstation troubleshoot-host]$ ansible servera.lab.example.com \
> -u devops -b -a "head /etc/postfix/main.cf"
servera.lab.example.com | FAILED | rc=1 >>
head: cannot open '/etc/postfix/main.cf' for reading: No such file or
directorynon-zero return code
```

Der Befehl ist fehlgeschlagen, weil das Playbook im Check-Modus ausgeführt wurde. Postfix ist auf `servera.lab.example.com` nicht installiert

- 4. Führen Sie das Playbook erneut aus, aber dieses Mal nicht im Check-Modus. Der Fehler in der Aufgabe *email notification of always\_bcc config* sollte nicht mehr vorhanden sein.

```
[student@workstation troubleshoot-host]$ ansible-playbook mailrelay.yml
PLAY [create mail relay servers] ****
...output omitted...
TASK [check for always_bcc] ****
changed: [servera.lab.example.com]

TASK [email notification of always_bcc config] ****
skipping: [servera.lab.example.com]

RUNNING HANDLER [restart postfix] ****
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=8    changed=5    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
```

- 5. Zeigen Sie mit einem Ad-hoc-Befehl den Anfang der Datei `/etc/postfix/main.cf` an.

```
[student@workstation troubleshoot-host]$ ansible servera.lab.example.com \
> -u devops -b -a "head /etc/postfix/main.cf"
servera.lab.example.com | SUCCESS | rc=0 >>
# Ansible managed
#
# Global Postfix configuration file. This file lists only a subset
```

**Kapitel 8 |** Fehlerbehebung in Ansible

```
# of all parameters. For the syntax, and for a complete parameter
# list, see the postconf(5) manual page (command: "man 5 postconf").
#
# For common configuration examples, see BASIC_CONFIGURATION_README
# and STANDARD_CONFIGURATION_README. To find these documents, use
# the command "postconf html_directory readme_directory", or go to
# http://www.postfix.org/BASIC_CONFIGURATION_README.html etc.
```

Jetzt beginnt die Datei mit einer Zeile, die die Zeichenfolge „Ansible managed“ enthält. Diese Datei wurde aktualisiert und wird jetzt von Ansible verwaltet.

- 6. Bearbeiten Sie das Playbook `mailrelay.yml`, und fügen Sie eine Aufgabe hinzu, um den `smtp`-Dienst über die Firewall zu aktivieren. Fügen Sie die Aufgabe als letzte Aufgabe vor den Handlern hinzu.

```
...output omitted...
- name: postfix firewalld config
  firewalld:
    state: enabled
    permanent: true
    immediate: true
    service: smtp
...output omitted...
```

- 7. Führen Sie das Playbook aus. Die Aufgabe `postfix firewalld config` sollte fehlerfrei ausgeführt worden sein.

```
[student@workstation troubleshoot-host]$ ansible-playbook mailrelay.yml
PLAY [create mail relay servers] ****
...output omitted...
TASK [postfix firewalld config] ****
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com      : ok=8      changed=2      unreachable=0      failed=0
skipped=1      rescued=0      ignored=0
```

- 8. Überprüfen Sie mit einem Ad-hoc-Befehl, ob der `smtp`-Service jetzt für die Firewall unter `servera.lab.example.com` konfiguriert ist.

```
[student@workstation troubleshoot-host]$ ansible servera.lab.example.com \
> -u devops -b -a "firewall-cmd --list-services"
servera.lab.example.com | CHANGED | rc=0 >>
cockpit dhcpcv6-client samba smtp ssh
```

- 9. Testen Sie mit `telnet`, ob der SMTP-Service den Port `TCP/25` unter „`servera.lab.example.com`“ überwacht. Trennen Sie die Verbindung, wenn Sie fertig sind.

```
[student@workstation troubleshoot-host]$ telnet servera.lab.example.com 25
Trying 172.25.250.10...
Connected to servera.lab.example.com.
Escape character is '^]'.
220 servera.lab.example.com ESMTP Postfix
quit
221 2.0.0 Bye
Connection closed by foreign host.
```

## Beenden

Führen Sie auf `workstation` das Skript `lab troubleshoot-host finish` aus, um diese Übung zu bereinigen.

```
[student@workstation ~]$ lab troubleshoot-host finish
```

Hiermit ist die angeleitete Übung beendet.

## ► Praktische Übung

# Fehlerbehebung in Ansible

### Performance-Checkliste

In dieser Übung werden Sie Probleme beheben, die bei dem Versuch auftreten, ein für Sie bereitgestelltes Playbook auszuführen.

### Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Fehler in Playbooks beheben.
- Fehler auf verwalteten Hosts beheben.

### Bevor Sie Beginnen

Melden Sie sich als student mit dem Passwort student bei workstation an. Führen Sie den Befehl `lab troubleshoot-review start` aus.

```
[student@workstation ~]$ lab troubleshoot-review start
```

Dieses Skript stellt sicher, dass Ansible auf workstation installiert ist, und erstellt das Verzeichnis `~student/troubleshoot-review/html/`. Es lädt die Dateien `ansible.cfg`, `inventory-lab`, `secure-web.yml` und `vhosts.conf` von `http://materials.example.com/labs/troubleshoot-review/` in das Verzeichnis `/home/student/troubleshoot-review/` herunter. Es lädt auch die Datei `index.html` in das Verzeichnis `/home/student/troubleshoot-review/html/` herunter.

### Anweisungen

1. Prüfen Sie im Verzeichnis `~/troubleshoot-review` die Syntax des Playbooks `secure-web.yml`. Dieses Playbook enthält ein Play, mit dem Apache HTTPD mit TLS/SSL für Hosts in der Gruppe `webserver` eingerichtet wird. Beheben Sie das gemeldete Problem.
2. Prüfen Sie die Syntax des Playbooks `secure-web.yml` erneut. Beheben Sie das gemeldete Problem.
3. Prüfen Sie die Syntax des Playbooks `secure-web.yml` ein drittes Mal. Beheben Sie das gemeldete Problem.
4. Prüfen Sie die Syntax des Playbooks `secure-web.yml` ein viertes Mal. Es sollten keine Syntaxfehler angezeigt werden.
5. Führen Sie das Playbook `secure-web.yml` aus. Ansible kann keine Verbindung mit `serverb.lab.example.com` herstellen. Beheben Sie dieses Problem.
6. Führen Sie das Playbook `secure-web.yml` erneut aus. Ansible sollte sich auf dem verwalteten Host als Remote-Benutzer `devops` authentifizieren. Beheben Sie dieses Problem.
7. Führen Sie das Playbook `secure-web.yml` ein drittes Mal aus. Beheben Sie das gemeldete Problem.

8. Führen Sie das Playbook `secure-web.yml` noch einmal aus. Es sollte erfolgreich abgeschlossen werden. Verwenden Sie einen Ad-hoc-Befehl, um zu verifizieren, dass der `httpd`-Service aktiv ist.

## Bewertung

Führen Sie auf `workstation` das Skript `lab troubleshoot-review grade` aus, um den Erfolg dieser Übung zu bestätigen.

```
[student@workstation ~]$ lab troubleshoot-review grade
```

## Beenden

Führen Sie auf `workstation` das Skript `lab troubleshoot-review finish` aus, um diese praktische Übung zu bereinigen.

```
[student@workstation ~]$ lab troubleshoot-review finish
```

Hiermit ist die praktische Übung abgeschlossen.

## ► Lösung

# Fehlerbehebung in Ansible

### Performance-Checkliste

In dieser Übung werden Sie Probleme beheben, die bei dem Versuch auftreten, ein für Sie bereitgestelltes Playbook auszuführen.

### Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Fehler in Playbooks beheben.
- Fehler auf verwalteten Hosts beheben.

### Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an. Führen Sie den Befehl `lab troubleshoot-review start` aus.

```
[student@workstation ~]$ lab troubleshoot-review start
```

Dieses Skript stellt sicher, dass Ansible auf `workstation` installiert ist, und erstellt das Verzeichnis `~student/troubleshoot-review/html/`. Es lädt die Dateien `ansible.cfg`, `inventory-lab`, `secure-web.yml` und `vhosts.conf` von `http://materials.example.com/labs/troubleshoot-review/` in das Verzeichnis `/home/student/troubleshoot-review/` herunter. Es lädt auch die Datei `index.html` in das Verzeichnis `/home/student/troubleshoot-review/html/` herunter.

### Anweisungen

1. Prüfen Sie im Verzeichnis `~/troubleshoot-review` die Syntax des Playbooks `secure-web.yml`. Dieses Playbook enthält ein Play, mit dem Apache HTTPD mit TLS/SSL für Hosts in der Gruppe `webserver` eingerichtet wird. Beheben Sie das gemeldete Problem.

- 1.1. Wechseln Sie auf `workstation` zum Projektverzeichnis `/home/student/troubleshoot-review`.

```
[student@workstation ~]$ cd ~/troubleshoot-review/
```

- 1.2. Prüfen Sie die Syntax des Playbooks `secure-web.yml`. Mit diesem Playbook wird Apache HTTPD mit TLS/SSL für Hosts in der Gruppe `webserver` eingerichtet, wenn keine Fehler aufgetreten sind.

```
[student@workstation troubleshoot-review]$ ansible-playbook --syntax-check \
> secure-web.yml
ERROR! We were unable to read either as JSON nor YAML, these are the errors we got
from each:
JSON: Expecting value: line 1 column 1 (char 0)
```

```
Syntax Error while loading YAML.  
  mapping values are not allowed in this context  
  
The error appears to be in '/home/student/troubleshoot-review/secure-web.yml':  
  line 7, column 30, but may be elsewhere in the file depending on the exact syntax  
problem.  
  
The offending line appears to be:  
  
vars:  
  random_var: This is colon: test  
          ^ here
```

- 1.3. Korrigieren Sie den Syntaxfehler in der Definition der Variablen `random_var`, indem Sie der Zeichenfolge `This is colon: test` doppelte Anführungszeichen hinzufügen. Die resultierende Änderung sollte wie folgt aussehen:

```
...output omitted...  
vars:  
  random_var: "This is colon: test"  
...output omitted...
```

2. Prüfen Sie die Syntax des Playbooks `secure-web.yml` erneut. Beheben Sie das gemeldete Problem.
  - 2.1. Prüfen Sie die Syntax von `secure-web.yml` erneut mit `ansible-playbook --syntax-check`.

```
[student@workstation troubleshoot-review]$ ansible-playbook --syntax-check \  
> secure-web.yml  
ERROR! We were unable to read either as JSON nor YAML, these are the errors we got  
from each:  
JSON: Expecting value: line 1 column 1 (char 0)  
  
Syntax Error while loading YAML.  
  did not find expected '-' indicator
```

```
The error appears to be in '/home/student/troubleshoot-review/secure-web.yml':  
  line 38, column 10, but may be elsewhere in the file depending on the exact  
syntax problem.
```

The offending line appears to be:

```
- name: start and enable web services  
  ^ here
```

- 2.2. Korrigieren Sie alle Syntaxfehler in der Einrückung. Entfernen Sie die überflüssigen Leerzeichen am Anfang der Elemente für die Aufgabe `start and enable web services`. Die resultierende Änderung sollte wie folgt aussehen:

```
...output omitted...
args:
  creates: /etc/pki/tls/certs/serverb.lab.example.com.crt

  - name: start and enable web services
    service:
      name: httpd
      state: started
      enabled: yes

  - name: deliver content
    copy:
      dest: /var/www/vhosts/serverb-secure
      src: html/
...output omitted...
```

3. Prüfen Sie die Syntax des Playbooks `secure-web.yml` ein drittes Mal. Beheben Sie das gemeldete Problem.

3.1. Prüfen Sie die Syntax des Playbooks `secure-web.yml`.

```
[student@workstation troubleshoot-review]$ ansible-playbook --syntax-check \
> secure-web.yml
ERROR! We were unable to read either as JSON nor YAML, these are the errors we got
from each:
JSON: Expecting value: line 1 column 1 (char 0)

Syntax Error while loading YAML.
  found unacceptable key (unhashable type: 'AnsibleMapping')

The error appears to be in '/home/student/troubleshoot-review/secure-web.yml':
  line 13, column 20, but may
  be elsewhere in the file depending on the exact syntax problem.
```

The offending line appears to be:

```
yum:
  name: {{ item }}
        ^ here
```

We could be wrong, but this one looks like it might be an issue with missing quotes. Always quote template expression brackets when they start a value. For instance:

```
with_items:
  - {{ foo }}
```

Should be written as:

```
with_items:
  - "{{ foo }}"
```

- 3.2. Korrigieren Sie in der Aufgabe `install web server packages` die Variable `item`. Fügen Sie `{} {{ item }}` doppelte Anführungszeichen hinzu. Die resultierende Änderung sollte wie folgt aussehen:

```
...output omitted...
  - name: install web server packages
    yum:
      name: "{{ item }}"
      state: latest
      notify:
        - restart services
    loop:
      - httpd
      - mod_ssl
...output omitted...
```

4. Prüfen Sie die Syntax des Playbooks `secure-web.yml` ein viertes Mal. Es sollten keine Syntaxfehler angezeigt werden.
- 4.1. Prüfen Sie die Syntax des Playbooks `secure-web.yml`. Es sollten keine Syntaxfehler angezeigt werden.

```
[student@workstation troubleshoot-review]$ ansible-playbook --syntax-check \
> secure-web.yml

playbook: secure-web.yml
```

5. Führen Sie das Playbook `secure-web.yml` aus. Ansible kann keine Verbindung mit `serverb.lab.example.com` herstellen. Beheben Sie dieses Problem.
- 5.1. Führen Sie das Playbook `secure-web.yml` aus. Dabei wird ein Fehler auftreten.

```
[student@workstation troubleshoot-review]$ ansible-playbook secure-web.yml

PLAY [create secure web service] ****
TASK [Gathering Facts] ****
fatal: [serverb.lab.example.com]: UNREACHABLE! => {"changed": false, "msg": "Failed to connect to the host via ssh: students@serverc.lab.example.com: Permission denied (publickey,gssapi-keyex,gssapi-with-mic,password).", "unreachable": true}

PLAY RECAP ****
serverb.lab.example.com : ok=0     changed=0     unreachable=1    failed=0 ...
```

- 5.2. Führen Sie das Playbook `secure-web.yml` erneut aus, wobei Sie den Parameter `-vvv` hinzufügen, um die Ausführlichkeit der Ausgabe zu erhöhen.  
Beachten Sie, dass Ansible die Verbindung anscheinend mit `serverc.lab.example.com` anstatt mit `serverb.lab.example.com` herstellt.

```
[student@workstation troubleshoot-review]$ ansible-playbook secure-web.yml -vvv
...output omitted...
TASK [Gathering Facts] ****
task path: /home/student/troubleshoot-review/secure-web.yml:3
<serverc.lab.example.com> ESTABLISH SSH CONNECTION FOR USER: students
<serverc.lab.example.com> SSH: EXEC ssh -C -o ControlMaster=auto
-o ControlPersist=60s -o KbdInteractiveAuthentication=no -o
PreferredAuthentications=gssapi-with-mic,gssapi-keyex,hostbased,publickey
-o PasswordAuthentication=no -o 'User="students"' -o ConnectTimeout=10 -o
ControlPath=/home/student/.ansible/cp/bc0c05136a serverc.lab.example.com '/bin/sh
-c """'echo ~students && sleep 0''''
...output omitted...
```

- 5.3. Korrigieren Sie die Zeile in der Datei `inventory`. Löschen Sie die Hostvariable `ansible_host`, sodass die Datei wie folgt angezeigt wird:

```
[webservers]
serverb.lab.example.com
```

6. Führen Sie das Playbook `secure-web.yml` erneut aus. Ansible sollte sich auf dem verwalteten Host als Remote-Benutzer `devops` authentifizieren. Beheben Sie dieses Problem.

- 6.1. Führen Sie das Playbook `secure-web.yml` aus.

```
[student@workstation troubleshoot-review]$ ansible-playbook secure-web.yml -vvv
...output omitted...
TASK [Gathering Facts] ****
task path: /home/student/troubleshoot-review/secure-web.yml:3
<serverb.lab.example.com> ESTABLISH SSH CONNECTION FOR USER: students
...output omitted...
fatal: [serverb.lab.example.com]: UNREACHABLE! => {
...output omitted...
```

- 6.2. Bearbeiten Sie das Playbook `secure-web.yml`, um sicherzustellen, dass `devops` `remote_user` für das Play darstellt. Die ersten Zeilen des Playbooks sollten wie folgt aussehen:

```
---
# start of secure web server playbook
- name: create secure web service
  hosts: webservers
  remote_user: devops
...output omitted...
```

7. Führen Sie das Playbook `secure-web.yml` ein drittes Mal aus. Beheben Sie das gemeldete Problem.

- 7.1. Führen Sie das Playbook `secure-web.yml` aus.

```
[student@workstation troubleshoot-review]$ ansible-playbook secure-web.yml -vvv
...output omitted...
failed: [serverb.lab.example.com] (item=mod_ssl) => {
    "ansible_loop_var": "item",
    "changed": false,
    "invocation": {
        "module_args": {
            "allow_downgrade": false,
            "autoremove": false,
...output omitted...
            "validate_certs": true
        }
    },
    "item": "mod_ssl",
    "msg": "This command has to be run under the root user.",
    "results": []
}
...output omitted...
```

- 7.2. Bearbeiten Sie das Play, um sicherzustellen, dass `become: true` oder `become: yes` festgelegt ist. Die resultierende Änderung sollte wie folgt aussehen:

```
---
# start of secure web server playbook
- name: create secure web service
  hosts: webservers
  remote_user: devops
  become: true
...output omitted...
```

8. Führen Sie das Playbook `secure-web.yml` noch einmal aus. Es sollte erfolgreich abgeschlossen werden. Verwenden Sie einen Ad-hoc-Befehl, um zu verifizieren, dass der `httpd`-Service aktiv ist.

- 8.1. Führen Sie das Playbook `secure-web.yml` aus.

```
[student@workstation troubleshoot-review]$ ansible-playbook secure-web.yml

PLAY [create secure web service] ****
...output omitted...
TASK [install web server packages] ****
changed: [serverb.lab.example.com] => (item=httpd)
changed: [serverb.lab.example.com] => (item=mod_ssl)
...output omitted...
TASK [httpd_conf_syntax variable] ****
ok: [serverb.lab.example.com] => {
    "msg": "The httpd_conf_syntax variable value is {'cmd': ['/sbin/httpd',
    '-t'], 'stdout': '', 'stderr': 'Syntax OK', 'rc': 0, 'start': '2021-07-16
    14:08:35.304347', 'end': '2021-07-16 14:08:35.342415', 'delta': '0:00:00.038068',
    'changed': True, 'stdout_lines': [], 'stderr_lines': ['Syntax OK'], 'failed':
    False, 'failed_when_result': False}"
}
...output omitted...
```

## Kapitel 8 | Fehlerbehebung in Ansible

```
RUNNING HANDLER [restart services] ****
changed: [serverb.lab.example.com]

PLAY RECAP ****
serverb.lab.example.com : ok=10    changed=7    unreachable=0    failed=0 ...
```

- 8.2. Ermitteln Sie mit einem Ad-hoc-Befehl den Status des `httpd`-Services auf `serverb.lab.example.com`. Der `httpd`-Service sollte jetzt auf `serverb.lab.example.com` ausgeführt werden.

```
[student@workstation troubleshoot-review]$ ansible all -u devops -b \
> -m command -a 'systemctl status httpd'
serverb.lab.example.com | CHANGED | rc=0 >>
● httpd.service - The Apache HTTP Server
  Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset:
  disabled)
  Active: active (running) since Fri 2021-07-16 14:08:37 EDT; 3min 1s ago
...output omitted...
```

## Bewertung

Führen Sie auf `workstation` das Skript `lab troubleshoot-review grade` aus, um den Erfolg dieser Übung zu bestätigen.

```
[student@workstation ~]$ lab troubleshoot-review grade
```

## Beenden

Führen Sie auf `workstation` das Skript `lab troubleshoot-review finish` aus, um diese praktische Übung zu bereinigen.

```
[student@workstation ~]$ lab troubleshoot-review finish
```

Hiermit ist die praktische Übung abgeschlossen.

# Zusammenfassung

---

In diesem Kapitel wurden die folgenden Themen behandelt:

- Ansible enthält eine integrierte Protokollierung. Diese Funktion ist nicht standardmäßig aktiviert.
- Der Parameter `log_path` im Abschnitt `default` der Konfigurationsdatei `ansible.cfg` legt den Speicherort der Protokolldatei fest, in die die gesamte Ansible-Ausgabe umgeleitet wird.
- Das Modul `debug` liefert bei der Ausführung eines Playbooks zusätzliche Debugging-Informationen (z. B. den aktuellen Wert einer Variable).
- Die Option `-v` des Befehls `ansible-playbook` bietet verschiedene Ausführlichkeitsebenen für die Ausgabe. Dies ist für das Debuggen von Ansible-Aufgaben beim Ausführen eines Playbooks nützlich.
- Die Option `--check` aktiviert Ansible-Module mit der Unterstützung für den Check-Modus, um auszuführende Änderungen anzuzeigen, anstatt diese Änderungen für die verwalteten Hosts vorzunehmen.
- Mit Ad-hoc-Befehlen können weitere Überprüfungen auf verwalteten Hosts ausgeführt werden.



## Kapitel 9

# Automatisieren von Linux-Administrationsaufgaben

### Ziel

Automatisierung der gängigen Linux-Systemadministrationsaufgaben mit Ansible.

### Ziele

- Systeme abonnieren, Softwarekanäle und Repositorys konfigurieren und RPM-Pakete auf verwalteten Hosts verwalten.
- Linux-Benutzer und -Gruppen verwalten, SSH konfigurieren und die Sudo-Konfiguration auf verwalteten Hosts ändern.
- Start eines Services verwalten, Prozesse mit at, cron und systemd planen, einen Reboot durchführen und das Standardstartziel auf verwalteten Hosts steuern.
- Speichergeräte partitionieren, LVM konfigurieren, Partitionen oder logische Volumes formatieren, Dateisysteme mounten und Swap-Dateien oder -Bereiche hinzufügen.
- Netzwerkeinstellungen und Namensauflösung für verwaltete Hosts konfigurieren und netzwerkbezogene Ansible-Fakten sammeln.

### Abschnitte

- Verwalten von Software und Subskriptionen (angeleitete Übung)
- Verwalten und Authentifizieren von Benutzern (angeleitete Übung)
- Verwalten des Bootvorgangs und geplanter Vorgänge (angeleitete Übung)
- Verwalten von Storage (angeleitete Übung)
- Verwalten der Netzwerkkonfiguration (angeleitete Übung)

### Praktische Übung

- Automatisieren von Linux-Administrationsaufgaben

# Verwalten von Software und Subskriptionen

---

## Ziele

Am Ende dieses Abschnitts sollten Sie in der Lage sein, Systeme zu abonnieren, Softwarekanäle und Repositorys zu konfigurieren, Modul-Streams zu aktivieren und RPM-Pakete auf verwalteten Hosts zu verwalten.

## Verwalten von Paketen mit Ansible

Das Ansible-Modul `yum` verwendet den *Yum-Paket-Manager* auf den verwalteten Hosts, um die Paketoperationen abzuwickeln. Das folgende Beispiel ist ein Playbook, das das Paket `httpd` auf dem von `servera.lab.example.com` verwalteten Host installiert.

```
---
- name: Install the required packages on the web server
  hosts: servera.lab.example.com
  tasks:
    - name: Install the httpd packages
      yum:
        name: httpd      ①
        state: present  ②
```

- ① Das Keyword `name` gibt den Namen des zu installierenden Pakets an.
- ② Das Keyword `state` gibt den erwarteten Status des Pakets auf dem verwalteten Host an:

### `present`

Ansible installiert das Paket, falls es noch nicht vorhanden ist.

### `absent`

Ansible entfernt das Paket, wenn es installiert ist.

### `latest`

Ansible aktualisiert das Paket, wenn es nicht bereits die aktuellste verfügbare Version aufweist. Wenn das Paket nicht installiert ist, installiert Ansible es.

In der folgenden Tabelle wird die Verwendung des Ansible-Moduls `yum` mit dem äquivalenten `yum`-Befehl verglichen.

Ansible-Aufgabe	Yum-Befehl
<pre>- name: Install httpd   yum:     name: httpd     state: present</pre>	<code>yum install httpd</code>

Ansible-Aufgabe	Yum-Befehl
<pre data-bbox="303 240 784 367"> - name: Install or update httpd   yum:     name: httpd     state: latest </pre>	<p data-bbox="850 240 1348 325">yum update httpd oder yum install httpd, wenn das Paket noch nicht installiert ist.</p>
<pre data-bbox="303 439 719 566"> - name: Update all packages   yum:     name: '*'     state: latest </pre>	<p data-bbox="850 424 1005 451">yum update</p>
<pre data-bbox="303 616 613 743"> - name: Remove httpd   yum:     name: httpd     state: absent </pre>	<p data-bbox="850 601 1095 629">yum remove httpd</p>
<pre data-bbox="303 815 809 941"> - name: Install Development Tools   yum:     name: '@Development Tools' ①     state: present </pre> <p data-bbox="303 988 817 1148">① Beim Ansible-Modul yum müssen Sie Gruppennamen @ voranstellen. Denken Sie daran, dass Sie die Liste der Gruppen mit dem Befehl yum group list abrufen können.</p>	<p data-bbox="850 800 1310 863">yum group install "Development Tools"</p>
<pre data-bbox="303 1201 793 1328"> - name: Remove Development Tools   yum:     name: '@Development Tools'     state: absent </pre>	<p data-bbox="850 1186 1295 1250">yum group remove "Development Tools"</p>

Ansible-Aufgabe	Yum-Befehl
<pre data-bbox="257 234 763 397"> - name: Inst perl AppStream   module   yum:     name: '@perl:5.26/minimal' ①     state: present </pre> <p data-bbox="257 439 747 760">① Um ein Yum AppStream-Modul zu verwalten, stellen Sie dem Namen @ voran. Die Syntax ist die gleiche wie beim Befehl yum. Sie können beispielsweise den Profilteil weglassen, um das Standardprofil zu verwenden: @perl:5.26. Denken Sie daran, dass Sie die verfügbaren Yum AppStream-Module mit dem Befehl yum module list auflisten können.</p>	<pre data-bbox="801 234 1241 287">yum module install perl:5.26/ minimal</pre>

Führen Sie den Befehl `ansible-doc yum` aus, um zusätzliche Parameter und Playbook-Beispiele abzurufen.

## Optimieren der Installation mehrerer Pakete

Um mehrere Pakete zu bearbeiten, akzeptiert das Keyword `name` eine Liste. Das folgende Beispiel zeigt ein Playbook, das drei Pakete auf `servera.lab.example.com` installiert.

```

---
- name: Install the required packages on the web server
  hosts: servera.lab.example.com
  tasks:
    - name: Install the packages
      yum:
        name:
          - httpd
          - mod_ssl
          - httpd-tools
      state: present

```

Mit dieser Syntax installiert Ansible die Pakete in einer einzigen Yum-Transaktion. Dies entspricht der Ausführung des Befehls `yum install httpd mod_ssl httpd-tools`.

Eine häufig verwendete, aber weniger effiziente und langsamere Version dieser Aufgabe ist die Verwendung einer Schleife.

```

---
- name: Install the required packages on the web server
  hosts: servera.lab.example.com
  tasks:
    - name: Install the packages
      yum:
        name: "{{ item }}"
        state: present

```

```
loop:
  - httpd
  - mod_ssl
  - httpd-tools
```

Vermeiden Sie diese Methode, da das Modul drei einzelne Transaktionen ausführen muss, eine für jedes Paket.

## Sammeln von Fakten zu installierten Paketen

Das Ansible-Modul `package_facts` sammelt die installierten Paketdetails auf verwalteten Hosts. Es legt die Variable `ansible_facts.packages` mit den Paketdetails fest.

Das folgende Playbook ruft das Modul `package_facts`, das Modul `debug` zum Anzeigen des Inhalts der Variablen `ansible_facts.packages` und erneut das Modul `debug` auf, um die Version des installierten `NetworkManager`-Pakets anzuzeigen.

```
---
- name: Display installed packages
  hosts: servera.lab.example.com
  tasks:
    - name: Gather info on installed packages
      package_facts:
        manager: auto

    - name: List installed packages
      debug:
        var: ansible_facts.packages

    - name: Display NetworkManager version
      debug:
        msg: "Version {{ansible_facts.packages['NetworkManager'][0].version}}"
        when: "'NetworkManager' in ansible_facts.packages"
```

Beim Ausführen zeigt das Playbook die Paketliste und die Version des `NetworkManager`-Pakets an:

```
[user@controlnode ~]$ ansible-playbook lspackages.yml

PLAY [Display installed packages] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Gather info on installed packages] ****
ok: [servera.lab.example.com]

TASK [List installed packages] ****
ok: [servera.lab.example.com] => {
  "ansible_facts.packages": {
    "NetworkManager": [
      {
        "arch": "x86_64",
        "epoch": 1,
        "name": "NetworkManager",
```

```

        "release": "14.el8",
        "source": "rpm",
        "version": "1.14.0"
    }
],
...output omitted...
"zlib": [
{
    "arch": "x86_64",
    "epoch": null,
    "name": "zlib",
    "release": "10.el8",
    "source": "rpm",
    "version": "1.2.11"
}
]
}
}

TASK [Display NetworkManager version] ****
ok: [servera.lab.example.com] => {
    "msg": "Version 1.14.0"
}

PLAY RECAP ****
servera.lab.example.com : ok=4     changed=0     unreachable=0     failed=0

```

## Überprüfen alternativer Module zum Verwalten von Paketen

Das Ansible-Modul `yum` funktioniert auf verwalteten Hosts, die den Yum-Paket-Manager verwenden. Für andere Paket-Manager stellt Ansible normalerweise ein dediziertes Modul bereit. Beispielsweise verwaltet das Modul `dnf` Pakete auf Betriebssystemen wie Fedora unter Verwendung des *DNF-Paket-Managers*. Das Modul `apt` verwendet das *APT-Paket-Tool*, das unter Debian oder Ubuntu verfügbar ist. Das Modul `win_package` kann Software auf Microsoft Windows-Systemen installieren.

Das folgende Playbook verwendet Bedingungen, um das entsprechende Modul in einer Umgebung auszuwählen, die aus Red Hat Enterprise Linux- und Fedora-Systemen besteht.

```

---
- name: Install the required packages on the web servers
  hosts: webservers
  tasks:
    - name: Install httpd on RHEL
      yum:
        name: httpd
        state: present
      when: "ansible_distribution == 'RedHat'"
    - name: Install httpd on Fedora
      dnf:
        name: httpd
        state: present
      when: "ansible_distribution == 'Fedora'"

```

Als Alternative erkennt und verwendet das generische Modul `package` automatisch den Paket-Manager, der auf den verwalteten Hosts verfügbar ist. Mit dem Modul `package` können Sie das vorherige Playbook wie folgt umschreiben.

```
---
- name: Install the required packages on the web servers
  hosts: webservers
  tasks:
    - name: Install httpd
      package:
        name: httpd
        state: present
```

Beachten Sie jedoch, dass das Modul `package` nicht alle Funktionen der spezielleren Module unterstützt. Außerdem haben Betriebssysteme häufig unterschiedliche Namen für die von ihnen bereitgestellten Pakete. Das Paket, mit dem der Apache-HTTP-Server installiert wird, lautet beispielsweise `httpd` unter Red Hat Enterprise Linux und `apache2` unter Ubuntu. In diesem Fall benötigen Sie noch eine Bedingung für die Auswahl des richtigen Paketnamens in Abhängigkeit vom Betriebssystem des verwalteten Hosts.

## Registrieren und Verwalten von Systemen mit der Red Hat-Subskriptionsverwaltung

Um Ihre neuen Red Hat Enterprise Linux-Systeme für Produktsubskriptionen zu berechtigen, bietet Ansible die Module `redhat_subscription` und `rhsml_repository`. Diese Module sind mit dem Tool für die *Red Hat-Subskriptionsverwaltung* auf den verwalteten Hosts gekoppelt.

### Registrieren und Abonnieren neuer Systeme

Die ersten beiden Aufgaben, die Sie normalerweise mit dem Tool für die Red Hat-Subskriptionsverwaltung ausführen, bestehen darin, das neue System zu registrieren und eine verfügbare Subskription anzuhängen.

Ohne Ansible führen Sie diese Aufgaben mit dem Befehl `subscription-manager` aus:

```
[user@host ~]$ subscription-manager register --username=yourusername \
> --password=yourpassword
[user@host ~]$ subscription-manager attach --pool=poolID
```

Denken Sie daran, dass Sie die verfügbaren Pools in Ihrem Konto mit dem Befehl `subscription-manager list --available` auflisten.

Das Ansible-Modul `redhat_subscription` führt die Registrierung und Subskription in einer Aufgabe aus.

```
- name: Register and subscribe the system
  redhat_subscription:
    username: yourusername
    password: yourpassword
    pool_ids: poolID
    state: present
```

Ein auf `present` festgelegtes `state`-Keyword weist darauf hin, dass das System registriert und abonniert werden muss. Wenn `absent` festgelegt ist, hebt das Modul die Registrierung des Systems auf.

## Aktivieren von Red Hat-Software-Repositorys

Die nächste Aufgabe nach der Subskription besteht darin, Red Hat-Software-Repositorys auf dem neuen System zu aktivieren.

Ohne Ansible führen Sie normalerweise den Befehl `subscription-manager` für diesen Zweck aus:

```
[user@host ~]$ subscription-manager repos \
> --enable "rhel-8-for-x86_64-baseos-rpms" \
> --enable "rhel-8-for-x86_64-baseos-debug-rpms"
```

Denken Sie daran, dass Sie die verfügbaren Repositorys mit dem Befehl `subscription-manager repos --list` auflisten können.

Verwenden Sie bei Ansible das Modul `rhsm_repository`:

```
- name: Enable Red Hat repositories
  rhsm_repository:
    name:
      - rhel-8-for-x86_64-baseos-rpms
      - rhel-8-for-x86_64-baseos-debug-rpms
    state: present
```

## Konfigurieren eines Yum-Repositorys

Um ein Drittanbieter-Repository auf einem verwalteten Host zu unterstützen, bietet Ansible das Modul `yum_repository`.

### Deklarieren eines Yum-Repositorys

Beim Ausführen deklariert das folgende Playbook ein neues Repository für `servera.lab.example.com`.

```
---
- name: Configure the company Yum repositories
  hosts: servera.lab.example.com
  tasks:
    - name: Ensure Example Repo exists
      yum_repository:
        file: example ①
        name: example-internal
        description: Example Inc. Internal YUM repo
        baseurl: http://materials.example.com/yum/repository/
        enabled: yes
        gpgcheck: yes ②
        state: present ③
```

- ❶ Das Keyword `file` gibt den Namen der unter dem Verzeichnis `/etc/yum.repos.d/` zu erstellenden Datei an. Das Modul fügt automatisch die Erweiterung `.repo` zu diesem Namen hinzu.
- ❷ Normalerweise signieren Softwareanbieter RPM-Pakete digital mit GPG-Schlüsseln. Durch Festlegen des Keyword `gpgcheck` auf `yes` verifiziert das RPM-System die Paketintegrität, indem es bestätigt, dass das Paket mit dem entsprechenden GPG-Schlüssel signiert wurde. Es lehnt die Installation eines Pakets ab, wenn die GPG-Signatur nicht übereinstimmt. Verwenden Sie das Ansible-Modul `rpm_key`, das später beschrieben wird, um den erforderlichen öffentlichen GPG-Schlüssel zu installieren.
- ❸ Wenn Sie das Keyword `state` auf `present` festlegen, erstellt oder aktualisiert Ansible die Datei `.repo`. Wenn `state` auf `absent` festgelegt ist, löscht Ansible die Datei.

Die resultierende Datei `/etc/yum.repos.d/example.repo` auf `servera.lab.example.com` sieht wie folgt aus.

```
[example-internal]
baseurl = http://materials.example.com/yum/repository/
enabled = 1
gpgcheck = 1
name = Example Inc. Internal YUM repo
```

Das Modul `yum_repository` stellt die meisten der Konfigurationsparameter des Yum-Repositorys als Keywords bereit. Führen Sie den Befehl `ansible-doc yum_repository` aus, um zusätzliche Parameter und Playbook-Beispiele abzurufen.



### Anmerkung

Manche Drittanbieter-Repositories enthalten die Konfigurationsdatei und den öffentlichen GPG-Schlüssel als Bestandteil eines RPM-Pakets, das mithilfe des Befehls `yum install` heruntergeladen und installiert werden kann. Beispielsweise stellt das Projekt *Extra Packages for Enterprise Linux (EPEL)* das Paket `https://dl.fedoraproject.org/pub/epel/epel-release-latest-VER.noarch.rpm` mit der Konfigurationsdatei `/etc/yum.repos.d/epel.repo` bereit. Verwenden Sie für dieses Repository das Ansible-Modul `yum` zur Installation des EPEL-Pakets anstelle des Moduls `yum_repository`.

## Importieren eines RPM-GPG-Schlüssels

Wenn das Keyword `gpgcheck` im Modul `yum_repository` auf `yes` festgelegt wird, müssen Sie auch den GPG-Schlüssel auf dem verwalteten Host installieren. Das Modul `rpm_key` im folgenden Beispiel stellt auf `servera.lab.example.com` den öffentlichen GPG-Schlüssel bereit, der auf einem Remote-Webserver gehostet wird.

```
---
- name: Configure the company Yum repositories
  hosts: servera.lab.example.com
  tasks:
    - name: Deploy the GPG public key
      rpm_key:
        key: http://materials.example.com/yum/repository/RPM-GPG-KEY-example
        state: present
```

```
- name: Ensure Example Repo exists
  yum_repository:
    file: example
    name: example-internal
    description: Example Inc. Internal YUM repo
    baseurl: http://materials.example.com/yum/repository/
    enabled: yes
    gpgcheck: yes
    state: present
```



## Literaturhinweise

Manpages `yum(8)`, `yum.conf(5)` und `subscription-manager(8)`

**yum – Manages packages with the yum package manager – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/modules/yum\\_module.html](https://docs.ansible.com/ansible/2.9/modules/yum_module.html)

**package\_facts – package information as facts – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/modules/package\\_facts\\_module.html](https://docs.ansible.com/ansible/2.9/modules/package_facts_module.html)

**redhat\_subscription – Manage registration and subscriptions to RHSM using the subscription-manager command – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/modules/redhat\\_subscription\\_module.html](https://docs.ansible.com/ansible/2.9/modules/redhat_subscription_module.html)

**rhsm\_repository – Manage RHSM repositories using the subscription-manager command – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/modules/rhsm\\_repository\\_module.html](https://docs.ansible.com/ansible/2.9/modules/rhsm_repository_module.html)

**yum\_repository – Add or remove YUM repositories – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/modules/yum\\_repository\\_module.html](https://docs.ansible.com/ansible/2.9/modules/yum_repository_module.html)

**rpm\_key – Adds or removes a gpg key from the rpm db – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/modules/rpm\\_key\\_module.html](https://docs.ansible.com/ansible/2.9/modules/rpm_key_module.html)

## ► Angeleitete Übung

# Verwalten von Software und Subskriptionen

In dieser Übung konfigurieren Sie ein neues Yum-Repository und installieren darin enthaltene Pakete auf Ihren verwalteten Hosts.

## Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Konfigurieren eines Yum-Repositorys mit dem Modul `yum_repository`.
- Verwalten von RPM-GPG-Schlüsseln mit dem Modul `rpm_key`.
- Abrufen von Informationen zu installierten Paketen auf einem Host mit dem Modul `package_facts`.

## Bevor Sie Beginnen

Führen Sie auf `workstation` das Start-Skript für die praktische Übung aus, um zu prüfen, ob die Umgebung für die praktische Übung bereit ist. Das Skript erstellt das Arbeitsverzeichnis `system-software` und füllt es mit einer Ansible-Konfigurationsdatei, dem Hostinventar und Dateien für die praktische Übung.

```
[student@workstation ~]$ lab system-software start
```

## Anweisungen

Ihre Organisation erfordert, dass auf allen Hosts das Paket `example-motd` installiert ist. Dieses Paket wird von einem internen Yum-Repository bereitgestellt, das von Ihrer Organisation verwaltet wird, um intern entwickelte Softwarepakete zu hosten.

Sie müssen ein Playbook schreiben, um sicherzustellen, dass das Paket `example-motd` auf dem Remote-Host installiert ist. Das Playbook muss die Konfiguration des internen Yum-Repositorys gewährleisten.

Das Repository befindet sich unter <http://materials.example.com/yum/repository>. Alle RPM-Pakete werden mit einem GPG-Schlüsselpaar der Organisation signiert. Der öffentliche GPG-Schlüssel befindet sich unter <http://materials.example.com/yum/repository/RPM-GPG-KEY-example>.

- 1. Wechseln Sie als Benutzer `student` auf `workstation` in das Arbeitsverzeichnis `/home/student/system-software`.

```
[student@workstation ~]$ cd ~/system-software  
[student@workstation system-software]$
```

- 2. Beginnen Sie mit dem Schreiben des Playbooks `repo_playbook.yml`. Definieren Sie ein einzelnes Play im Playbook, das auf alle Hosts ausgerichtet ist. Fügen Sie eine `vars`-

Klausel hinzu, die eine einzelne `custom_pkg`-Variable mit dem Wert `example-motd` definiert. Fügen Sie die Klausel `tasks` zum Playbook hinzu.

Das Playbook enthält jetzt Folgendes:

```
---
```

```
- name: Repository Configuration
  hosts: all
  vars:
    custom_pkg: example-motd
  tasks:
```

► **3.** Fügen Sie zwei Aufgaben zum Playbook hinzu.

Sammeln Sie mit dem Modul `package_facts` in der ersten Aufgabe Informationen zu den auf dem Remote-Host installierten Paketen. Diese Aufgabe füllt der Fakt `ansible_facts.packages` aus.

Verwenden Sie das Modul `debug` in der zweiten Aufgabe, um die installierte Version des Pakets zu drucken, auf das von der Variablen `custom_pkg` verwiesen wird. Führen Sie diese Aufgabe nur aus, wenn das benutzerdefinierte Paket im Fakt `ansible_facts.packages` enthalten ist.

Führen Sie das Playbook `repo_playbook.yml` aus.

- 3.1. Fügen Sie die erste Aufgabe zum Playbook hinzu. Konfigurieren Sie das Keyword `manager` des Moduls `package_facts` mit einem Wert von `auto`. Die erste Aufgabe enthält Folgendes:

```
- name: Gather Package Facts
  package_facts:
    manager: auto
```

- 3.2. Fügen Sie dem Playbook eine zweite Aufgabe hinzu, die das Modul `debug` zum Anzeigen des Werts der Variablen `ansible_facts.packages[custom_pkg]` verwendet. Fügen Sie eine `when`-Klausel zur Aufgabe hinzu, um zu prüfen, ob der Wert der Variablen `custom_pkg` in der Variablen `ansible_facts.packages` enthalten ist. Die zweite Aufgabe enthält Folgendes:

```
- name: Show Package Facts for the custom package
  debug:
    var: ansible_facts.packages[custom_pkg]
  when: custom_pkg in ansible_facts.packages
```

- 3.3. Führen Sie das folgende Playbook aus:

```
[student@workstation system-software]$ ansible-playbook repo_playbook.yml
```

```
PLAY [Repository Configuration] ****
```

```
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]
```

```
TASK [Gather Package Facts] ****
ok: [servera.lab.example.com]
```

```
TASK [Show Package Facts for the custom package] ****
skipping: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com    : ok=2      changed=0      unreachable=0      failed=0
skipped=1      rescued=0      ignored=0
```

Die Debug-Aufgabe wird übersprungen, da das Paket `example-motd` nicht auf dem Remote-Host installiert ist.

- 4. Fügen Sie eine dritte Aufgabe hinzu, die das Modul `yum_repository` verwendet, um die Konfiguration des internen Yum-Repositorys auf dem Remote-Host zu gewährleisten. Stellen Sie Folgendes sicher:

- Die Konfiguration des Repositorys wird in der Datei `/etc/yum.repos.d/example.repo` gespeichert.
- Die Repository-ID lautet `example-internal`.
- Die Basis-URL lautet `http://materials.example.com/yum/repository`.
- Das Repository ist für die Überprüfung von RPM-GPG-Signaturen konfiguriert.
- Die Beschreibung des Repositorys lautet `Example Inc. Internal YUM repo`.

Die dritte Aufgabe enthält Folgendes:

```
- name: Ensure Example Repo exists
  yum_repository:
    name: example-internal
    description: Example Inc. Internal YUM repo
    file: example
    baseurl: http://materials.example.com/yum/repository/
    gpgcheck: yes
```

- 5. Fügen Sie die vierte Aufgabe zum Play hinzu, die das Modul `rpm_key` verwendet, um sicherzustellen, dass der öffentliche Schlüssel des Repositorys auf dem Remote-Host vorhanden ist. Die URL des öffentlichen Schlüssels für das Repository lautet `http://materials.example.com/yum/repository/RPM-GPG-KEY-example`.

Die vierte Aufgabe sieht wie folgt aus:

```
- name: Ensure Repo RPM Key is Installed
  rpm_key:
    key: http://materials.example.com/yum/repository/RPM-GPG-KEY-example
    state: present
```

- 6. Fügen Sie eine fünfte Aufgabe hinzu, um sicherzustellen, dass das Paket, auf das die Variable `custom_pkg` verweist, auf dem Remote-Host installiert ist.

Die fünfte Aufgabe sieht wie folgt aus:

```
- name: Install Example motd package
  yum:
    name: "{{ custom_pkg }}"
    state: present
```

- 7. Der Fakt `ansible_facts.packages` wird nicht aktualisiert, wenn ein neues Paket auf einem Remote-Host installiert wird.

Kopieren Sie die zweite Aufgabe und fügen Sie sie als sechste Aufgabe zum Play hinzu. Führen Sie das Playbook aus und verifizieren Sie, dass der Fakt `ansible_facts.packages` keine Informationen zum Modul `example-motd` enthält, das auf dem Remote-Host installiert ist.

7.1. Die sechste Aufgabe enthält eine Kopie der zweiten Aufgabe:

```
- name: Show Package Facts for the custom package
  debug:
    var: ansible_facts.packages[custom_pkg]
  when: custom_pkg in ansible_facts.packages
```

Das gesamte Playbook sieht jetzt wie folgt aus:

```
---
- name: Repository Configuration
  hosts: all
  vars:
    custom_pkg: example-motd
  tasks:
    - name: Gather Package Facts
      package_facts:
        manager: auto

    - name: Show Package Facts for the custom package
      debug:
        var: ansible_facts.packages[custom_pkg]
      when: custom_pkg in ansible_facts.packages

    - name: Ensure Example Repo exists
      yum_repository:
        name: example-internal
        description: Example Inc. Internal YUM repo
        file: example
        baseurl: http://materials.example.com/yum/repository/
        gpgcheck: yes

    - name: Ensure Repo RPM Key is Installed
      rpm_key:
        key: http://materials.example.com/yum/repository/RPM-GPG-KEY-example
        state: present
```

```
- name: Install Example motd package
  yum:
    name: "{{ custom_pkg }}"
    state: present

- name: Show Package Facts for the custom package
  debug:
    var: ansible_facts.packages[custom_pkg]
  when: custom_pkg in ansible_facts.packages
```

## 7.2. Führen Sie das Playbook aus.

```
[student@workstation system-software]$ ansible-playbook repo_playbook.yml
PLAY [Repository Configuration] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Gather Package Facts] ****
ok: [servera.lab.example.com] ①

TASK [Show Package Facts for the custom package] ****
skipping: [servera.lab.example.com]

TASK [Ensure Example Repo exists] ****
changed: [servera.lab.example.com]

TASK [Ensure Repo RPM Key is Installed] ****
changed: [servera.lab.example.com]

TASK [Install Example motd package] ****
changed: [servera.lab.example.com]

TASK [Show Package Facts for the custom package] ****
skipping: [servera.lab.example.com] ②

PLAY RECAP ****
servera.lab.example.com      : ok=5      changed=3      unreachable=0      failed=0
skipped=2      rescued=0      ignored=0
```

- ① Die Aufgabe zum Sammeln von Paketfakten ermittelt die Daten, die im Fakt `ansible_facts.packages` enthalten sind.
- ② Die Aufgabe wird übersprungen, da das Paket `example-motd` nach der Aufgabe zum Sammeln von Paketfakten installiert wird.

- 8. Fügen Sie eine Aufgabe unmittelbar nach der Aufgabe zum Installieren des `motd`-Beispieldatums mit dem Modul `package_facts` hinzu, um die Paketfakten zu aktualisieren. Legen Sie für das `manager`-Keyword des Moduls den Wert `auto` fest.

Das vollständige Playbook wird unten angezeigt:

```
---
- name: Repository Configuration
  hosts: all
  vars:
    custom_pkg: example-motd
  tasks:
    - name: Gather Package Facts
      package_facts:
        manager: auto

    - name: Show Package Facts for the custom package
      debug:
        var: ansible_facts.packages[custom_pkg]
      when: custom_pkg in ansible_facts.packages

    - name: Ensure Example Repo exists
      yum_repository:
        name: example-internal
        description: Example Inc. Internal YUM repo
        file: example
        baseurl: http://materials.example.com/yum/repository/
        gpgcheck: yes

    - name: Ensure Repo RPM Key is Installed
      rpm_key:
        key: http://materials.example.com/yum/repository/RPM-GPG-KEY-example
        state: present

    - name: Install Example motd package
      yum:
        name: "{{ custom_pkg }}"
        state: present

    - name: Gather Package Facts
      package_facts:
        manager: auto

    - name: Show Package Facts for the custom package
      debug:
        var: ansible_facts.packages[custom_pkg]
      when: custom_pkg in ansible_facts.packages
```

- 9. Verwenden Sie einen Ad-hoc-Befehl von Ansible, um das während der vorherigen Ausführung des Playbooks installierte Paket *example-motd* zu entfernen. Führen Sie das Playbook mit der eingefügten Aufgabe `package_facts` aus und verifizieren Sie anhand der Ausgabe die Installation des Pakets *example-motd*.
- 9.1. Um das Paket *example-motd* von allen Hosts zu entfernen, verwenden Sie den Befehl `ansible all` mit den Optionen `-m yum` und `-a 'name=example-motd state=absent'`.

```
[student@workstation system-software]$ ansible all -m yum \
> -a 'name=example-motd state=absent'
servera.lab.example.com | CHANGED => {
...output omitted...
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
        "Removed: example-motd-1.0-1.el7.x86_64"
    ]
...output omitted...
```

## 9.2. Führen Sie das Playbook aus.

```
[student@workstation system-software]$ ansible-playbook repo_playbook.yml

PLAY [Repository Configuration] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Gather Package Facts] ****
ok: [servera.lab.example.com]

TASK [Show Package Facts for the custom package] ****
skipping: [servera.lab.example.com] ①

...output omitted...

TASK [Install Example motd package] ****
changed: [servera.lab.example.com] ②

TASK [Gather Package Facts] ****
ok: [servera.lab.example.com] ③

TASK [Show Package Facts for example-motd] ****
ok: [servera.lab.example.com] => {
    "ansible_facts.packages[custom_pkg)": [④
        {
            "arch": "x86_64",
            "epoch": null,
            "name": "example-motd",
            "release": "1.el7",
            "source": "rpm",
            "version": "1.0"
        }
    ]
}

PLAY RECAP ****
servera.lab.example.com      : ok=7      changed=1      unreachable=0      failed=0
skipped=1      rescued=0      ignored=0
```

- ❶ Für das Paket `example-motd` ist kein Paketfakt vorhanden, da das Paket nicht auf dem Remote-Host installiert ist.
- ❷ Das Paket `example-motd` wird als Ergebnis dieser Aufgabe installiert, was durch den Status `changed` erkennbar ist.
- ❸ Diese Aufgabe aktualisiert die Paketfakten mit Informationen zum Paket `example-motd`.
- ❹ Der `example-motd`-Paketfakt ist vorhanden und zeigt an, dass nur ein `example-motd`-Paket installiert ist. Das installierte Paket weist die Version `1.0` auf.

## Beenden

Führen Sie auf `workstation` das Skript `lab system-software finish` aus, um die in dieser Übung erstellten Ressourcen zu bereinigen.

```
[student@workstation ~]$ lab system-software finish
```

Hiermit ist die angeleitete Übung beendet.

# Verwalten von Benutzern und der Authentifizierung

---

## Ziele

Nach Abschluss dieses Abschnitts sollten Sie in der Lage sein, Linux-Benutzer und -Gruppen zu verwalten, SSH zu konfigurieren und die Sudo-Konfiguration auf verwalteten Hosts zu ändern.

## Modul „user“

Mit dem Ansible-Modul „user“ können Sie Benutzerkonten auf einem Remote-Host verwalten. Sie können eine Reihe von Parametern verwalten, einschließlich Benutzer entfernen, Benutzerverzeichnis festlegen, die UID für Systemkonten festlegen, Kennwörter und zugehörige Gruppierungen verwalten. Um einen Benutzer zu erstellen, der sich am Computer anmelden kann, müssen Sie ein gehashtes Passwort für den Kennwortparameter angeben. Im Referenzabschnitt finden Sie einen Link zu „Wie erstelle ich verschlüsselte Kennwörter für das Modul „user“?“

### Beispiel für das Modul „user“

```
- name: Add new user to the development machine and assign the appropriate groups.
  user:
    name: devops_user ①
    shell: /bin/bash ②
    groups: sys_admins, developers ③
    append: yes
```

- ①** Der Parameter `name` ist die einzige Anforderung im Modul „user“ und normalerweise das Service- oder Benutzerkonto.
- ②** Der Parameter `shell` legt optional die Shell des Benutzers fest. Auf anderen Betriebssystemen wird die Standard-Shell vom verwendeten Tool festgelegt.
- ③** Der Parameter `groups` mit dem Parameter `append` informiert den Rechner, dass wir die Gruppen „sys\_asmins“ und „developers“ mit diesem Benutzer anhängen möchten. Wenn Sie den Parameter „append“ nicht verwenden, werden die Gruppen überschrieben.

Beim Erstellen eines Benutzers können Sie `generate_ssh_key` angeben. Dies überschreibt einen vorhandenen SSH-Schlüssel nicht.

### Beispiel für das Modul „user“, das einen SSH-Schlüssel generiert

```
- name: Create a SSH key for user1
  user:
    name: user1
    generate_ssh_key: yes
    ssh_key_bits: 2048
    ssh_key_file: .ssh/id_my_rsa
```

**Anmerkung**

Das Modul „user“ bietet auch einige Rückgabewerte. Ansible-Module können einen Rückgabewert in einer Variable registrieren. Erfahren Sie mehr in der Ansible-Dokumentation und auf der Hauptdokumentationsseite.

**Einige häufig verwendete Parameter**

Parameter	Kommentare
comment	Legt optional die Beschreibung eines Benutzerkontos fest.
group	Legt optional die primäre Gruppe des Benutzers fest.
groups	Liste mehrerer Gruppen. Wenn ein Nullwert festgelegt ist, werden alle Gruppen mit Ausnahme der primären Gruppe entfernt.
home	Legt optional das Benuterverzeichnis fest.
create_home	Hat einen Booleschen Wert von „yes“ oder „no“. Ein Benuterverzeichnis wird für den Benutzer erstellt, wenn der Wert „yes“ ist.
system	Beim Erstellen eines Kontos mit state=present und Festlegung auf „yes“ wird das Benutzer- zu einem Systemkonto. Diese Einstellung kann für vorhandene Benutzer nicht geändert werden.
uid	Legt die UID des Benutzers fest.

**Modul „group“**

Mit dem Modul group können Sie Gruppen auf den verwalteten Hosts verwalten (hinzufügen, löschen, ändern). Sie benötigen groupadd, groupdel oder groupmod. Verwenden Sie für Windows-Ziele das Modul win\_group.

**Beispiel für das Modul „group“**

```
- name: Verify that auditors group exists
  group:
    name: auditors
    state: present
```

**Parameter für das Modul „group“**

Parameter	Kommentare
gid	Optionale GID, die für die Gruppe festgelegt werden soll.
local	Erzwingt die Verwendung von "lokalen" Befehsalternativen auf Plattformen, die diese implementieren.

Parameter	Kommentare
name	Name der zu verwaltenden Gruppe.
state	Gibt an, ob die Gruppe auf dem Remote-Host vorhanden sein soll oder nicht.
system	„yes“ gibt an, dass die erstellte Gruppe eine Systemgruppe ist.

## Das Modul „known\_hosts“

Wenn Sie eine große Anzahl von Hostschlüsseln verwalten möchten, verwenden Sie das Modul `known_hosts`. Mit dem Modul `known_hosts` können Sie Host-Schlüssel aus der Datei „`known_hosts`“ auf einem verwalteten Host hinzufügen oder entfernen.

### Beispiel für `known_host`-Aufgaben

```
- name: copy host keys to remote servers
  known_hosts:
    path: /etc/ssh/ssh_known_hosts
    name: host1
    key: "{{ lookup('file', 'pubkeys/host1') }}"
```

- ❶ Ein `lookup`-Plugin ermöglicht es Ansible, auf Daten von externen Quellen zuzugreifen.

## Modul „authorized\_key“

Mit dem Modul `authorized_key` können Sie autorisierte SSH-Schlüssel pro Benutzerkonto hinzufügen oder entfernen. Wenn Sie Benutzer zu einer großen Anzahl von Servern hinzufügen und entfernen, müssen Sie in der Lage sein, SSH-Schlüssel zu verwalten.

### Beispiel für `authorized_key`-Aufgaben

```
- name: Set authorized key
  authorized_key:
    user: user1
    state: present
    key: "{{ lookup('file', '/home/user1/.ssh/id_rsa.pub') }}"
```

- ❶ Ein Schlüssel kann auch aus einer URL entnommen werden: <https://github.com/user1.keys>.



## Literaturhinweise

### Users Module – Ansible-Dokumentation

[http://docs.ansible.com/ansible/2.9/modules/user\\_module.html#user-module](http://docs.ansible.com/ansible/2.9/modules/user_module.html#user-module)

### Wie erstelle ich verschlüsselte Kennwörter für das Modul „user“?

[https://docs.ansible.com/ansible/2.9/reference\\_appendices/faq.html#how-do-i-generate-encrypted-passwords-for-the-user-module](https://docs.ansible.com/ansible/2.9/reference_appendices/faq.html#how-do-i-generate-encrypted-passwords-for-the-user-module)

### Modul „group“ – Ansible-Dokumentation

[https://docs.ansible.com/ansible/2.9/modules/group\\_module.html#group-module](https://docs.ansible.com/ansible/2.9/modules/group_module.html#group-module)

### Modul „known\_hosts“ (SSH) – Ansible-Dokumentation

[https://docs.ansible.com/ansible/2.9/modules/known\\_hosts\\_module.html#known-hosts-module](https://docs.ansible.com/ansible/2.9/modules/known_hosts_module.html#known-hosts-module)

### Modul „authorized key“ – Ansible-Dokumentation

[https://docs.ansible.com/ansible/2.9/modules/authorized\\_key\\_module.html#authorized-key-module](https://docs.ansible.com/ansible/2.9/modules/authorized_key_module.html#authorized-key-module)

### lookup-Plugin – Ansible-Dokumentation

<https://docs.ansible.com/ansible/2.9/plugins/lookup.html?highlight=lookup>

## ► Angeleitete Übung

# Verwalten von Benutzern und der Authentifizierung

In dieser Übung erstellen Sie mehrere Benutzer auf Ihren verwalteten Hosts und füllen die autorisierten SSH-Schlüssel für sie.

## Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Erstellen einer neuen Benutzergruppe.
- Verwalten von Benutzern mit dem Modul `user`.
- Füllen von autorisierten SSH-Schlüsseln mit dem Modul `authorized_key`.
- Ändern der Dateien `sudoers` und `sshd_config` mit dem Modul `lineinfile`.

## Bevor Sie Beginnen

Führen Sie auf `workstation` das Übungs-Start-Skript aus, um zu prüfen, ob die Umgebung für die Übung bereit ist. Das Skript erstellt das Arbeitsverzeichnis `system-users` und füllt es mit einer Ansible-Konfigurationsdatei, einem Hostinventar und einigen Lab-Dateien.

```
[student@workstation ~]$ lab system-users start
```

## Anweisungen

Ihre Organisation erfordert, dass für alle Hosts dieselben lokalen Benutzer verfügbar sind. Diese Benutzer sollten zur Benutzergruppe `webadmin` gehören, die über die Möglichkeit verfügt, den Befehl `sudo` ohne Angabe eines Passworts zu verwenden. Zudem sollten die öffentlichen SSH-Schlüssel der Benutzer in der Umgebung verteilt sein und dem Benutzer `root` sollte es nicht gestattet sein, sich direkt mit SSH anzumelden.

Sie müssen ein Playbook schreiben, um sicherzustellen, dass die Benutzer und die Benutzergruppe auf dem Remote-Host vorhanden sind. Das Playbook muss sicherstellen, dass sich die Benutzer mit dem autorisierten SSH-Schlüssel anmelden und `sudo` ohne Angabe eines Passworts verwenden können. Zudem darf sich der Benutzer `root` nicht direkt mit SSH anmelden.

- 1. Wechseln Sie als Benutzer `student` auf `workstation` in das Arbeitsverzeichnis `/home/student/system-users`.

```
[student@workstation ~]$ cd ~/system-users
[student@workstation system-users]$
```

- 2. Betrachten Sie die vorhandene Variablendatei `vars/users_vars.yml`.

```
[student@workstation system-users]$ cat vars/users_vars.yml
---
users:
  - username: user1
    groups: webadmin
  - username: user2
    groups: webadmin
  - username: user3
    groups: webadmin
  - username: user4
    groups: webadmin
  - username: user5
    groups: webadmin
```

Sie verwendet den Variablenamen `username`, um den richtigen Benutzernamen festzulegen, und die Variable `groups`, um weitere Gruppen zu definieren, denen der Benutzer angehören soll.

- 3. Beginnen Sie mit dem Schreiben des Playbooks `users.yml`. Definieren Sie ein einzelnes Play im Playbook, das auf die Hostgruppe `webservers` ausgerichtet ist. Fügen Sie eine `vars_files`-Klausel hinzu, die den Speicherort des Dateinamens `vars/users_vars.yml` definiert, der für Sie erstellt wurde. Zudem enthält sie alle für diese Übung erforderlichen Benutzernamen. Fügen Sie die Klausel `tasks` zum Playbook hinzu. Verwenden Sie einen Texteditor, um das Playbook `users.yml` zu erstellen. Das Playbook sollte folgenden Inhalt aufweisen:

```
---
- name: Create multiple local users
  hosts: webservers
  vars_files:
    - vars/users_vars.yml
  tasks:
```

- 4. Fügen Sie zwei Aufgaben zum Playbook hinzu.

Verwenden Sie das Modul `group` in der ersten Aufgabe, um die Benutzergruppe `webadmin` auf dem Remote-Host zu erstellen. Mithilfe dieser Aufgabe wird die Gruppe `webadmin` erstellt.

Verwenden Sie das Modul `user` in der zweiten Aufgabe, um die Benutzer aus der Datei `vars/users_vars.yml` zu erstellen.

Führen Sie das Playbook `users.yml` aus.

- 4.1. Fügen Sie die erste Aufgabe zum Playbook hinzu. Die erste Aufgabe enthält Folgendes:

```
- name: Add webadmin group
  group:
    name: webadmin
    state: present
```

- 4.2. Fügen Sie dem Playbook eine zweite Aufgabe hinzu, die das Modul `user` zum Erstellen der Benutzer verwendet. Fügen Sie die Klausel `loop: "{{ users }}"`

zur Aufgabe hinzu, um die Variablenliste für jeden Benutzernamen in der Datei `vars/users_vars.yml` zu durchlaufen. Verwenden Sie als `name`: für die Benutzer `item.username` als Variablennamen. Auf diese Weise enthält die Variablenliste möglicherweise zusätzliche Informationen, die zum Erstellen der Benutzer hilfreich sein können, z. B. die Gruppen, denen die Benutzer angehören sollen. Die zweite Aufgabe enthält Folgendes:

```
- name: Create user accounts
  user:
    name: "{{ item.username }}"
    groups: "{{ item.groups }}"
  loop: "{{ users }}"
```

4.3. Führen Sie das folgende Playbook aus:

```
[student@workstation system-users]$ ansible-playbook users.yml

PLAY [Create multiple local users] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Add webadmin group] ****
changed: [servera.lab.example.com]

TASK [Create user accounts] ****
changed: [servera.lab.example.com] => (item={'username': 'user1', 'groups': 'webadmin'})
changed: [servera.lab.example.com] => (item={'username': 'user2', 'groups': 'webadmin'})
changed: [servera.lab.example.com] => (item={'username': 'user3', 'groups': 'webadmin'})
changed: [servera.lab.example.com] => (item={'username': 'user4', 'groups': 'webadmin'})
changed: [servera.lab.example.com] => (item={'username': 'user5', 'groups': 'webadmin'})

PLAY RECAP ****
servera.lab.example.com      : ok=3      changed=2      unreachable=0      failed=0
```

- 5. Fügen Sie eine dritte Aufgabe hinzu, die das Modul `authorized_key` verwendet, um sicherzustellen, dass die öffentlichen SSH-Schlüssel auf dem Remote-Host ordnungsgemäß verteilt wurden. Im Verzeichnis `files` verfügt jeder Benutzer über eine eindeutige öffentliche SSH-Schlüsseldatei. Das Modul durchläuft die Liste der Benutzer, findet den entsprechenden Schlüssel mithilfe der Variablen `username` und überträgt den Schlüssel per Pushvorgang an den Remote-Host.

Die dritte Aufgabe enthält Folgendes:

```
- name: Add authorized keys
  authorized_key:
    user: "{{ item.username }}"
    key: "{{ lookup('file', 'files/'+ item.username + '.key.pub') }}"
  loop: "{{ users }}"
```

- 6. Fügen Sie dem Play, das das Modul copy verwendet, eine vierte Aufgabe hinzu, um die Konfigurationsdatei sudo zu ändern, und gestatten Sie den Gruppenmitgliedern von webadmin die Verwendung von sudo ohne Passwort auf dem Remote-Host.

Die vierte Aufgabe sieht wie folgt aus:

```
- name: Modify sudo config to allow webadmin users sudo without a password
  copy:
    content: "%webadmin ALL=(ALL) NOPASSWD: ALL"
    dest: /etc/sudoers.d/webadmin
    mode: 0440
```

- 7. Fügen Sie eine fünfte Aufgabe hinzu, um sicherzustellen, dass sich der Benutzer root nicht direkt mit SSH anmelden kann. Verwenden Sie notify: "Restart sshd", um einen Handler auszulösen und SSH neu zu starten.

Die fünfte Aufgabe sieht wie folgt aus:

```
- name: Disable root login via SSH
  lineinfile:
    dest: /etc/ssh/sshd_config
    regexp: "^\$PermitRootLogin"
    line: "PermitRootLogin no"
  notify: Restart sshd
```

- 8. Fügen Sie in der ersten Zeile nach dem Speicherort der Variablendatei eine neue Handler-Definition hinzu. Weisen Sie dem Handler die Bezeichnung Restart sshd zu.

8.1. Der Handler sollte wie folgt definiert sein:

```
...output omitted...
- vars/users_vars.yml
handlers:
- name: Restart sshd
  service:
    name: sshd
    state: restarted
```

Das gesamte Playbook sieht jetzt wie folgt aus:

```
---
- name: Create multiple local users
  hosts: webservers
  vars_files:
    - vars/users_vars.yml
  handlers:
```

```
- name: Restart sshd
  service:
    name: sshd
    state: restarted

  tasks:

    - name: Add webadmin group
      group:
        name: webadmin
        state: present

    - name: Create user accounts
      user:
        name: "{{ item.username }}"
        groups: "{{ item.groups }}"
      loop: "{{ users }}"

    - name: Add authorized keys
      authorized_key:
        user: "{{ item.username }}"
        key: "{{ lookup('file', 'files/' + item.username + '.key.pub') }}"
      loop: "{{ users }}"

    - name: Modify sudo config to allow webadmin users sudo without a password
      copy:
        content: "%webadmin ALL=(ALL) NOPASSWD: ALL"
        dest: /etc/sudoers.d/webadmin
        mode: 0440

    - name: Disable root login via SSH
      lineinfile:
        dest: /etc/ssh/sshd_config
        regexp: "^\$PermitRootLogin"
        line: "PermitRootLogin no"
      notify: "Restart sshd"
```

## 8.2. Führen Sie das Playbook aus.

```
[student@workstation system-users]$ ansible-playbook users.yml

PLAY [Create multiple local users] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Add webadmin group] ****
ok: [servera.lab.example.com]

TASK [Create user accounts] ****
ok: [servera.lab.example.com] => (item={'username': 'user1', 'groups': 'webadmin'})
ok: [servera.lab.example.com] => (item={'username': 'user2', 'groups': 'webadmin'})
```

```
ok: [servera.lab.example.com] => (item={'username': u'username': u'groups': u'webadmin'})  
ok: [servera.lab.example.com] => (item={'username': u'username': u'groups': u'webadmin'})  
ok: [servera.lab.example.com] => (item={'username': u'username': u'groups': u'webadmin'})  
  
TASK [Add authorized keys] ****  
changed: [servera.lab.example.com] => (item={'username': u'username': u'groups': u'webadmin'})  
  
TASK [Modify sudo config to allow webadmin users sudo without a password] ***  
changed: [servera.lab.example.com]  
  
TASK [Disable root login via SSH] ****  
changed: [servera.lab.example.com]  
  
RUNNING HANDLER [Restart sshd] ****  
changed: [servera.lab.example.com]  
  
PLAY RECAP ****  
servera.lab.example.com : ok=7    changed=4    unreachable=0    failed=0
```

- 9. Melden Sie sich als Benutzer `user1` mit SSH auf dem Server `servera` an. Sobald Sie angemeldet sind, verwenden Sie den Befehl `sudo su -`, um die Identität in den Benutzer `root` zu ändern.

- 9.1. Verwenden Sie SSH als Benutzer `user1` und melden Sie sich auf dem Server `servera` an.

```
[student@workstation system-users]$ ssh user1@servera  
Activate the web console with: systemctl enable --now cockpit.socket  
  
[user1@servera ~]$
```

- 9.2. Ändern Sie die Identität in Benutzer `root`.

```
[user1@servera ~]$ sudo -i  
root@servera ~]#
```

- 9.3. Melden Sie sich von `servera` ab.

```
[root@servera ~]$ exit  
logout  
[user1@servera ~]$ exit  
logout  
Connection to servera closed.  
[student@workstation system-users]$
```

- **10.** Versuchen Sie, sich direkt als Benutzer `root` auf dem Server `servera` anzumelden. Bei diesem Schritt sollte ein Fehler auftreten, da die SSH-Daemon-Konfiguration so geändert wurde, dass keine direkten Benutzeranmeldungen von `root` gestattet werden.

- 10.1. Verwenden Sie auf `workstation` SSH als `root`, um sich auf dem Server `servera` anzumelden.

```
[student@workstation system-users]$ ssh root@servera  
root@servera's password: redhat  
Permission denied, please try again.  
root@servera's password:
```

Dies bestätigt, dass die SSH-Konfiguration für den Benutzer `root` den direkten Zugriff auf das System verweigert hat.

## Beenden

Führen Sie auf `workstation` das Skript `lab system-users finish` aus, um die in dieser Übung erstellten Ressourcen zu bereinigen.

```
[student@workstation ~]$ lab system-users finish
```

Hiermit ist die angeleitete Übung beendet.

# Verwalten des Bootvorgangs und geplanter Vorgänge

---

## Ziele

Am Ende dieses Abschnitts sollten Sie zu Folgendem in der Lage sein: Start eines Services verwalten, Prozesse mit at, cron und systemd planen, einen Reboot durchführen und das Standardstartziel auf verwalteten Hosts steuern.

## Zeitplanung mit dem Modul „at“

Die schnelle einmalige Zeitplanung erfolgt mit dem Modul at. Sie erstellen den Job für eine zukünftige Ausführungszeit und er wird bis zu dieser Ausführungszeit angehalten. Es gibt sechs Parameter für dieses Modul. Diese sind: „command“, „count“, „script\_file“, „state“, „unique“ und „units“.

### Beispiel für Modul „at“:

```
- name: remove tempuser.
  at:
    command: userdel -r tempuser
    count: 20
    units: minutes
    unique: yes
```

### Parameter

Parameter	Optionen	Kommentare
command	Null	Befehl, dessen Ausführung geplant ist.
count	Null	Anzahl der Einheiten. (Muss mit Einheiten ausgeführt werden)
script_file	Null	Vorhandene Skriptdatei, die in Zukunft ausgeführt werden soll.
state	absent, present	Der Status fügt einen Befehl oder ein Skript hinzu oder entfernt sie.
unique	yes, no	Wenn ein Job bereits ausgeführt wird, wird er nicht erneut ausgeführt.
units	minutes/hours/days/weeks	Zeitangaben.

## Anhängen von Befehlen mit dem Modul „cron“

Wenn Sie eine geplante Aufgabe für Jobs festlegen, wird das Modul cron verwendet. Das Modul „cron“ hängt Befehle direkt in der crontab des von Ihnen angegebenen Benutzers an.

### Beispiel für Modul „cron“:

```
- cron:  
  name: "Flush Bolt"  
  user: "root"  
  minute: 45  
  hour: 11  
  job: "php ./app/nut cache:clear"
```

Dieses Play verwendet den Befehl „cache:clear“ für ein Unternehmen. Der Bolt-Cache wird umgehend geleert, wobei Cache-Dateien und -Verzeichnisse entfernt werden. Der Cache des CMS-Servers wird jeden Morgen um 11:45 Uhr geleert.

Ansible schreibt das Play mit der vom Benutzer angegebenen korrekten Syntax in die crontab.

Wenn Sie die crontab überprüfen, wird verifiziert, ob an sie angehängt wurde.

Einige häufig verwendete Parameter für das Modul „cron“ sind:

#### Parameter

Parameter	Optionen	Kommentare
special_time	reboot, yearly, annually, monthly, weekly, daily, hourly	Reihe von wiederkehrenden Zeiten.
state	absent, present	Bei Auswahl von „present“ wird der Befehl erstellt. Bei Auswahl von „absent“ wird er entfernt.
cron_file	Null	Wenn Sie eine große Anzahl von Servern warten müssen, ist es manchmal besser, eine vorab erstellte crontab-Datei zu haben.
backup	yes, no	Sichert die crontab-Datei vor dem Bearbeiten.

## Verwalten von Services mit den Modulen „systemd“ und „service“

Für das Verwalten von Services oder das Neuladen von Daemons verfügt Ansible über die Module `systemd` und „`service`“. „`service`“ bietet eine Reihe grundlegender Optionen zum Starten, Stoppen, Neustarten und Aktivieren. Das Modul „`systemd`“ bietet weitere Konfigurationsoptionen. Mit „`systemd`“ können Sie im Gegensatz zum Modul „`service`“ einen Daemon neu laden.

## Beispiel für Modul „service“:

```
- name: start nginx
  service:
    name: nginx
    state: started"
```



### Anmerkung

Der Init-Daemon wird durch „systemd“ ersetzt. In vielen Fällen ist „systemd“ die bessere Option.

## Beispiel für Modul „systemd“:

```
- name: reload web server
  systemd:
    name: apache2
    state: reload
    daemon-reload: yes
```

## Modul „reboot“

Ein weiteres häufig genutztes Ansible-Systemmodul ist `reboot`. Es wird zum Initiieren des Herunterfahrens als sicherer angesehen als das Modul „shell“. Während der Ausführung eines Plays fährt das Modul "reboot" den verwalteten Host herunter und wartet, bis er wieder gestartet wurde, bevor das Play fortgesetzt wird.

## Beispiel für Modul „reboot“:

```
- name: "Reboot after patching"
  reboot:
    reboot_timeout: 180

- name: force a quick reboot
  reboot:
```

## Module „shell“ und „command“

Wie die Module „service“ und „systemd“ können „shell“ und „command“ einige Aufgaben austauschen. Das Modul „command“ gilt als sicherer, aber einige Umgebungsvariablen sind nicht verfügbar. Stream-Operatoren funktionieren auch nicht. Wenn Sie Ihre Befehle streamen müssen, reicht das Modul „shell“ aus.

## Beispiel für Modul „shell“:

```
- name: Run a templated variable (always use quote filter to avoid injection)
  shell: cat {{ myfile|quote }}❶
```

- ❶ Um alle Variablen zu bereinigen, wird empfohlen, dass Sie `{{ var | quote }}` statt `{{ var }}` verwenden.

## Beispiel für Modul „command“:

```
- name: This command only
  command: /usr/bin/scrape_logs.py arg1 arg2
  args:①
    chdir: scripts/
    creates: /path/to/script
```

- ① Sie können Argumente in das Formular übergeben, um die Optionen bereitzustellen.



### Anmerkung

Das Modul „command“ gilt als sicherer, da es von der Benutzerumgebung nicht betroffen ist.

Durch das Erfassen von Fakten auf dem verwalteten Host können Sie auf die Umgebungsvariablen zugreifen. Es gibt eine Unterliste namens „ansible\_env“, die alle Umgebungsvariablen beinhaltet.

```
---
- name:
  hosts: webservers
  vars:
    local_shell: "{{ ansible_env }}"①
  tasks:
    - name: Printing all the environment variables in Ansible
      debug:
        msg: "{{ local_shell }}"
```

- ① Sie können die Variable, die Sie zurückgeben möchten, mithilfe des lookup-Plugins isolieren.  
msg: "{{ lookup ('env', 'USER', 'HOME', 'SHELL') }}"



### Literaturhinweise

**at – Schedule the execution of a command or script file via the at command – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/modules/at\\_module.html](https://docs.ansible.com/ansible/2.9/modules/at_module.html)

**cron – Manage cron.d and crontab entries – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/modules/cron\\_module.html](https://docs.ansible.com/ansible/2.9/modules/cron_module.html)

**reboot – Reboot a machine – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/modules/reboot\\_module.html](https://docs.ansible.com/ansible/2.9/modules/reboot_module.html)

**service – Run services on a machine – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/modules/service\\_module.html](https://docs.ansible.com/ansible/2.9/modules/service_module.html)

## ► Angeleitete Übung

# Verwalten des Boot-Prozesses und geplanter Prozesse

In dieser Übung verwalten Sie den Startvorgang, planen wiederkehrende Jobs und starten verwaltete Hosts neu.

## Ergebnisse

Sie sollten ein Playbook für Folgendes verwenden können:

- Planen eines cron-Jobs.
- Entfernen eines einzelnen bestimmten cron-Jobs aus einer crontab-Datei.
- Planen einer at-Aufgabe.
- Das Standard-Bootziel auf verwalteten Hosts festlegen.
- Verwaltete Hosts neu booten.

## Bevor Sie Beginnen

Führen Sie das Skript `lab system-process start` über `workstation` aus, um die Umgebung für diese Übung zu konfigurieren. Das Skript erstellt das Arbeitsverzeichnis `system-process` und lädt die Ansible-Konfigurationsdatei und die Hostinventardatei herunter, die für die Übung erforderlich sind.

```
[student@workstation ~]$ lab system-process start
```

## Anweisungen

- 1. Wechseln Sie als Benutzer `student` auf `workstation` in das Arbeitsverzeichnis `/home/student/system-process`.

```
[student@workstation ~]$ cd ~/system-process
[student@workstation system-process]$
```

- 2. Erstellen Sie das Playbook `create_crontab_file.yml` im aktuellen Arbeitsverzeichnis. Konfigurieren Sie das Playbook für die Verwendung des Moduls `cron`, um die crontab-Datei `/etc/cron.d/add-date-time` zu erstellen, die eine wiederkehrende cron-Aufgabe plant. Der Job sollte alle zwei Minuten als Benutzer `devops` zwischen `09:00` und `16:59` Uhr von Montag bis Freitag ausgeführt werden. Der Job sollte das aktuelle Datum und die aktuelle Uhrzeit an die Datei „`/home/devops/my_datetime_cron_job`“ anhängen.
- 2.1. Erstellen Sie das neue Playbook `create_crontab_file.yml`, und fügen Sie die Zeilen hinzu, die für den Start des Plays benötigt werden. Es sollte die verwalteten Hosts in der Gruppe `webservers` als Ziel verwenden und die Rechteerweiterung aktivieren.

```
---
```

```
- name: Recurring cron job
  hosts: webservers
  become: true
```

- 2.2. Definieren Sie eine Aufgabe, die das Modul `cron` verwendet, um eine wiederkehrende Cron-Aufgabe zu planen.



### Anmerkung

Das Modul `cron` bietet eine `name`-Option, um den crontab-Dateieintrag eindeutig zu beschreiben und die erwarteten Ergebnisse sicherzustellen. Die Beschreibung wird der crontab-Datei hinzugefügt. Die Option `name` ist z. B. erforderlich, wenn Sie einen crontab-Eintrag mit `state=absent` entfernen. Darüber hinaus verhindert die Option `name`, dass ein neuer crontab-Eintrag erstellt wird, wenn der Standardstatus `state=present` festgelegt ist.

```
tasks:
  - name: Crontab file exists
    cron:
      name: Add date and time to a file
```

- 2.3. Konfigurieren Sie den Job so, dass er alle zwei Minuten zwischen 09:00 und 16:59 Uhr von Montag bis Freitag ausgeführt wird.

```
minute: "*/2"
hour: 9-16
weekday: 1-5
```

- 2.4. Verwenden Sie den Parameter `cron_file` für die Verwendung der crontab-Datei `/etc/cron.d/add-date-time` anstelle der crontab-Datei eines einzelnen Benutzers in `/var/spool/cron/`. Ein relativer Pfad positioniert die Datei im Verzeichnis `/etc/cron.d`. Wenn der Parameter `cron_file` verwendet wird, müssen Sie auch den Parameter `user` angeben.

```
user: devops
job: date >> /home/devops/my_date_time_cron_job
cron_file: add-date-time
state: present
```

- 2.5. Wenn Sie fertig sind, sollte das Playbook wie folgt aussehen: Überprüfen Sie das Playbook auf Fehlerfreiheit.

```
---
```

```
- name: Recurring cron job
  hosts: webservers
  become: true
```

```
tasks:
  - name: Crontab file exists
```

```
cron:  
  name: Add date and time to a file  
  minute: "*/2"  
  hour: 9-16  
  weekday: 1-5  
  user: devops  
  job: date >> /home/devops/my_date_time_cron_job  
  cron_file: add-date-time  
  state: present
```

- 2.6. Verifizieren Sie die Syntax des Playbooks, indem Sie den Befehl `ansible-playbook --syntax-check > create_crontab_file.yml` ausführen. Korrigieren Sie sämtliche Fehler, bevor Sie mit dem nächsten Schritt fortfahren.

```
[student@workstation system-process]$ ansible-playbook --syntax-check \  
> create_crontab_file.yml  
  
playbook: create_crontab_file.yml
```

- 2.7. Führen Sie das Playbook aus.

```
[student@workstation system-process]$ ansible-playbook create_crontab_file.yml  
  
PLAY [Recurring cron job] *****  
  
TASK [Gathering Facts] *****  
ok: [servera.lab.example.com]  
  
TASK [Crontab file exists] *****  
changed: [servera.lab.example.com]  
  
PLAY RECAP *****  
servera.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```

- 2.8. Führen Sie einen Ad-hoc-Befehl aus, um zu verifizieren, dass die cron-Datei `/etc/cron.d/add-date-time` vorhanden und der Inhalt korrekt ist.

```
[student@workstation system-process]$ ansible webservers -u devops -b \  
> -a "cat /etc/cron.d/add-date-time"  
servera.lab.example.com | CHANGED | rc=0 >>  
#Ansible: Add date and time to a file  
*/2 9-16 * * 1-5 devops date >> /home/devops/my_date_time_cron_job
```

- 3. Erstellen Sie das Playbook `remove_cron_job.yml` im aktuellen Arbeitsverzeichnis. Konfigurieren Sie das Playbook für die Verwendung des cron-Moduls, um den cron-Job `Add date and time to a file` über die crontab-Datei `/etc/cron.d/add-date-time` zu erstellen.
- 3.1. Erstellen Sie das neue Playbook `remove_cron_job.yml`, und fügen Sie die folgenden Zeilen hinzu:

```
---  
- name: Remove scheduled cron job  
  hosts: webservers  
  become: true  
  
  tasks:  
    - name: Cron job removed  
      cron:  
        name: Add date and time to a file  
        user: devops  
        cron_file: add-date-time  
        state: absent
```

- 3.2. Verifizieren Sie die Syntax des Playbooks, indem Sie den Befehl `ansible-playbook --syntax-check remove_cron_job.yml` ausführen. Korrigieren Sie sämtliche Fehler, bevor Sie mit dem nächsten Schritt fortfahren.

```
[student@workstation system-process]$ ansible-playbook --syntax-check \  
> remove_cron_job.yml  
  
playbook: remove_cron_job.yml
```

- 3.3. Führen Sie das Playbook aus.

```
[student@workstation system-process]$ ansible-playbook remove_cron_job.yml  
  
PLAY [Remove scheduled cron job] *****  
  
TASK [Gathering Facts] *****  
ok: [servera.lab.example.com]  
  
TASK [Cron job removed] *****  
changed: [servera.lab.example.com]  
  
PLAY RECAP *****  
servera.lab.example.com : ok=2     changed=1     unreachable=0     failed=0
```

- 3.4. Führen Sie einen Ad-hoc-Befehl aus, um zu verifizieren, dass die cron-Datei `/etc/cron.d/add-date-time` weiterhin vorhanden ist, der cron-Job jedoch entfernt wurde.

```
[student@workstation system-process]$ ansible webservers -u devops -b \  
> -a "cat /etc/cron.d/add-date-time"  
servera.lab.example.com | CHANGED | rc=0 >>
```

- 4. Erstellen Sie das Playbook `schedule_at_task.yml` im aktuellen Arbeitsverzeichnis. Konfigurieren Sie das Playbook für die Verwendung des Moduls `at`, um eine Aufgabe zu planen, die eine Minute in der Zukunft ausgeführt wird. Die Aufgabe sollte den Befehl `date` ausführen und seine Ausgabe an die Datei `/home/devops/my_at_date_time` weiterleiten. Verwenden Sie die Option `unique: yes`, um sicherzustellen, dass keine neue Aufgabe hinzugefügt wird, wenn der Befehl bereits in der `at`-Warteschlange vorhanden ist.

- 4.1. Erstellen Sie das neue Playbook `schedule_at_task.yml`, und fügen Sie die folgenden Zeilen hinzu:

```
---
- name: Schedule at task
  hosts: webservers
  become: true
  become_user: devops

  tasks:
    - name: Create date and time file
      at:
        command: "date > ~/my_at_date_time\n"
        count: 1
        units: minutes
        unique: yes
        state: present
```

- 4.2. Verifizieren Sie die Syntax des Playbooks, indem Sie den Befehl `ansible-playbook --syntax-check schedule_at_task.yml` ausführen. Korrigieren Sie sämtliche Fehler, bevor Sie mit dem nächsten Schritt fortfahren.

```
[student@workstation system-process]$ ansible-playbook --syntax-check \
> schedule_at_task.yml

playbook: schedule_at_task.yml
```

- 4.3. Führen Sie das Playbook aus.

```
[student@workstation system-process]$ ansible-playbook schedule_at_task.yml

PLAY [Schedule at task] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Create date and time file] ****
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com      : ok=2      changed=1      unreachable=0      failed=0
```

- 4.4. Nachdem Sie eine Minute auf den Abschluss des `at`-Befehls gewartet haben, führen Sie Ad-hoc-Befehle aus, um zu verifizieren, dass die Datei `/home/devops/my_at_date_time` vorhanden ist und den richtigen Inhalt enthält.

```
[student@workstation system-process]$ ansible webservers -u devops \
> -a "ls -l my_at_date_time"
servera.lab.example.com | CHANGED | rc=0 >>
-rw-rw-r--. 1 devops devops 30 abr 17 06:15 my_at_date_time

[student@workstation system-process]$ ansible webservers -u devops \
```

```
> -a "cat my_at_date_time"
servera.lab.example.com | CHANGED | rc=0 >
Thu Jul 22 13:24:34 PDT 2021
```

- 5. Erstellen Sie das Playbook `set_default_boot_target_graphical.yml` im aktuellen Arbeitsverzeichnis. Konfigurieren Sie das Playbook für die Verwendung des Moduls `file`, um den symbolischen Link auf verwalteten Hosts derart zu ändern, dass er auf das Bootziel `graphical-target` verweist.



#### Anmerkung

Im folgenden `file`-Modul verweist der symbolische Link auf den Parameterwert `src`. Der Parameterwert `dest` ist der symbolische Link.

- 5.1. Erstellen Sie das neue Playbook `set_default_boot_target_graphical.yml`, und fügen Sie die folgenden Zeilen hinzu:

```
---
- name: Change default boot target
  hosts: webservers
  become: true

  tasks:
    - name: Default boot target is graphical
      file:
        src: /usr/lib/systemd/system/graphical.target
        dest: /etc/systemd/system/default.target
        state: link
```

- 5.2. Verifizieren Sie die Syntax des Playbooks, indem Sie den Befehl `ansible-playbook --syntax-check set_default_boot_target_graphical.yml` ausführen. Korrigieren Sie sämtliche Fehler, bevor Sie mit dem nächsten Schritt fortfahren.

```
[student@workstation system-process]$ ansible-playbook --syntax-check \
> set_default_boot_target_graphical.yml

playbook: set_default_boot_target_graphical.yml
```

- 5.3. Führen Sie vor dem Ausführen des Playbooks einen Ad-hoc-Befehl aus, um zu verifizieren, dass das aktuelle Standard-Bootziel `multi-user.target` entspricht:

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "systemctl get-default"
servera.lab.example.com | CHANGED | rc=0 >
multi-user.target
```

- 5.4. Führen Sie das Playbook aus.

```
[student@workstation system-process]$ ansible-playbook \
> set_default_boot_target_graphical.yml

PLAY [Change default boot target] ****
```

```
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Default boot target is graphical] ****
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```

- 5.5. Führen Sie einen Ad-hoc-Befehl aus, um zu verifizieren, dass das Standard-Bootziel jetzt `graphical.target` ist.

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "systemctl get-default"
servera.lab.example.com | CHANGED | rc=0 >>
graphical.target
```

- 6. Erstellen Sie das Playbook `reboot_hosts.yml` im aktuellen Arbeitsverzeichnis, das die verwalteten Hosts neu bootet. Es ist nicht erforderlich, einen Server nach dem Ändern des Standardziels neu zu booten. Es kann sich jedoch als nützlich erweisen, zu wissen, wie ein Playbook erstellt wird, mit dem verwaltete Hosts neu gebootet werden.

- 6.1. Erstellen Sie das neue Playbook `reboot_hosts.yml`, und fügen Sie die folgenden Zeilen hinzu:

```
---
- name: Reboot hosts
  hosts: webservers
  become: true

  tasks:
    - name: Hosts are rebooted
      reboot:
```

- 6.2. Verifizieren Sie die Syntax des Playbooks, indem Sie den Befehl `ansible-playbook --syntax-check reboot_hosts.yml` ausführen. Korrigieren Sie sämtliche Fehler, bevor Sie mit dem nächsten Schritt fortfahren.

```
[student@workstation system-process]$ ansible-playbook --syntax-check \
> reboot_hosts.yml

playbook: reboot_hosts.yml
```

- 6.3. Führen Sie vor dem Ausführen des Playbooks einen Ad-hoc-Befehl aus, um den Zeitstempel für den letzten System-Reboot zu ermitteln.

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "who -b"
servera.lab.example.com | CHANGED | rc=0 >>
  system boot 2021-07-22 14:34
```

- 6.4. Führen Sie das Playbook aus.

```
[student@workstation system-process]$ ansible-playbook reboot_hosts.yml

PLAY [Reboot hosts] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Hosts are rebooted] ****
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```

- 6.5. Führen Sie einen Ad-hoc-Befehl aus, um den Zeitstempel für den letzten System-Reboot zu ermitteln. Der Zeitstempel, der nach der Ausführung des Playbooks angezeigt wird, sollte einen späteren Zeitpunkt aufweisen.

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "who -b"
servera.lab.example.com | CHANGED | rc=0 >>
    system boot 2021-07-22 14:52
```

- 6.6. Führen Sie einen zweiten Ad-hoc-Befehl aus, um zu ermitteln, ob das Bootziel graphical.target den Reboot überstanden hat.

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "systemctl get-default"
servera.lab.example.com | CHANGED | rc=0 >>
graphical.target
```

- 7. Um die Konsistenz in den verbleibenden Übungen zu erhalten, legen Sie für das Standard-Bootziel wieder die vorherige Einstellung, multi-user.target, fest. Erstellen Sie das Playbook set\_default\_boot\_target\_multi-user.yml im aktuellen Arbeitsverzeichnis. Konfigurieren Sie das Playbook für die Verwendung des Moduls file, um den symbolischen Link auf verwalteten Hosts so zu ändern, dass er auf das Bootziel multi-user.target verweist.
- 7.1. Erstellen Sie das neue Playbook set\_default\_boot\_target\_multi-user.yml, und fügen Sie die folgenden Zeilen hinzu:

```
---
- name: Change default runlevel target
  hosts: webservers
  become: true

  tasks:
    - name: Default runlevel is multi-user target
      file:
```

```
src: /usr/lib/systemd/system/multi-user.target
dest: /etc/systemd/system/default.target
state: link
```

- 7.2. Verifizieren Sie die Syntax des Playbooks, indem Sie den Befehl `ansible-playbook --syntax-check set_default_boot_target_multi-user.yml` ausführen. Korrigieren Sie sämtliche Fehler, bevor Sie mit dem nächsten Schritt fortfahren.

```
[student@workstation system-process]$ ansible-playbook --syntax-check \
> set_default_boot_target_multi-user.yml

playbook: set_default_boot_target_multi-user.yml
```

- 7.3. Führen Sie das Playbook aus.

```
[student@workstation system-process]$ ansible-playbook \
> set_default_boot_target_multi-user.yml

PLAY [Change default runlevel target] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Default runlevel is multi-user target] ****
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```

- 7.4. Führen Sie einen Ad-hoc-Befehl aus, um zu verifizieren, dass das Standard-Bootziel jetzt `multi-user.target` ist.

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "systemctl get-default"
servera.lab.example.com | CHANGED | rc=0 >>
multi-user.target
```

## Beenden

Führen Sie auf `workstation` das Skript `lab system-process finish` aus, um diese Übung zu bereinigen.

```
[student@workstation ~]$ lab system-process finish
```

Hiermit ist die angeleitete Übung beendet.

# Verwalten von Storage

## Ziele

Am Ende dieses Abschnitts sollten Sie zu Folgendem in der Lage sein: Speichergeräte partitionieren, LVM konfigurieren, Partitionen oder logische Volumes formatieren, Dateisysteme anhängen und Swap-Dateien oder -Bereiche hinzufügen.

## Konfigurieren des Storage mit Ansible-Modulen

Red Hat Ansible Automation Platform bietet eine Sammlung von Modulen zum Konfigurieren von Speichergeräten auf verwalteten Hosts. Diese Module unterstützen das Partitionieren von Geräten, das Erstellen logischer Volumes sowie das Erstellen und Mounten von Dateisystemen.

### Modul parted

Das Modul `parted` unterstützt die Partitionierung von Blockgeräten. Dieses Modul beinhaltet die Funktionalität des Befehls `parted` und ermöglicht das Erstellen von Partitionen mit einer bestimmten Größe, Markierung und Ausrichtung. In der folgenden Tabelle sind einige Parameter für das Modul `parted` aufgeführt.

Parametername	Beschreibung
align	Konfiguriert die Partitionsausrichtung.
device	Blockgerät.
flags	Flags für die Partition.
number	Die Partitionsnummer.
part_end	Partitionsgröße ab dem Anfang der Festplatte, die in von <code>parted</code> unterstützten Einheiten angegeben wird.
state	Erstellt oder entfernt die Partition.
unit	Größeneinheiten für die Partitionsinformationen.

Das folgende Beispiel erstellt eine neue Partition von 10 GB.

```
- name: New 10GB partition
  parted:
    device: /dev/vdb ①
    number: 1 ②
    state: present ③
    part_end: 10GB ④
```

**①** Verwendet `vdb` als zu partitionierendes Blockgerät.

**②** Erstellt die Partition 1.

- ③ Stellt sicher, dass die Partition verfügbar ist.
- ④ Legt die Partitionsgröße auf 10 GB fest.

## Module lvg und lvol

Die Module `lvg` und `lvol` unterstützen die Erstellung logischer Volumes, einschließlich der Konfiguration physischer Volumes und Volumegruppen. `lvg` verwendet als Parameter die Blockgeräte, um sie als physische Back-End-Volumes für die Volumegruppe zu konfigurieren. In der folgenden Tabelle sind einige Parameter für das Modul `lvg` aufgeführt.

Parametername	Beschreibung
<code>pesize</code>	Die Größe des physischen Extents. Muss eine Potenz von 2 oder ein Vielfaches von 128 KiB sein.
<code>pvs</code>	Liste von kommagetrennten Geräten, die als physische Volumes für die Volumegruppe konfiguriert werden sollen.
<code>vg</code>	Der Name der Volumegruppe.
<code>state</code>	Erstellt oder entfernt das Volume.

Mit der folgenden Aufgabe wird eine Volumegruppe mit einer bestimmten physischen Extentgröße erstellt, wobei ein Blockgerät als Back-End verwendet wird.

```
- name: Creates a volume group
  lvg:
    vg: vg1 ①
    pvs: /dev/vda1 ②
    pesize: 32 ③
```

- ① Der Name der Volumegruppe lautet `vg1`.
- ② Verwendet `/dev/vda1` als physisches Back-End-Volume für die Volumegruppe.
- ③ Legt die physische Extendgröße auf 32 fest.

Wenn im folgenden Beispiel die Volumegruppe `vg1` bereits mit `/dev/vdb1` als physischem Volume verfügbar ist, wird das Volume durch Hinzufügen eines neuen physischen Volumes mit `/dev/vdc1` vergrößert.

```
- name: Resize a volume group
  lvg:
    vg: vg1
    pvs: /dev/vdb1,/dev/vdc1
```

Das Modul `lvol` erstellt logische Volumes und unterstützt die Vergrößerung und Verkleinerung dieser Volumes sowie der darüber liegenden Dateisysteme. Dieses Modul unterstützt auch die Erstellung von Snapshots für die logischen Volumes. In der folgenden Tabelle sind einige Parameter für das Modul `lvol` aufgeführt.

Parametername	Beschreibung
<code>lv</code>	Der Name des logischen Volumes.

Parametername	Beschreibung
resizefs	Ändert die Größe des Dateisystems mit dem logischen Volume.
shrink	Ermöglicht die Verkleinerung des logische Volumes.
size	Die Größe des logischen Volumes.
snapshot	Der Name des Snapshots für das logische Volume.
state	Erstellt oder entfernt das logische Volume.
vg	Die übergeordnete Volumegruppe für das logische Volume.

Die folgende Aufgabe erstellt ein logisches Volume von 2 GB.

```
- name: Create a logical volume of 2GB
  lvol:
    vg: vg1 ①
    lv: lv1 ②
    size: 2g ③
```

- ① Der Name der übergeordneten Volumegruppe lautet vg1.
- ② Der Name des logischen Volumes lautet lv1.
- ③ Die Größe des logischen Volumes ist 2 GB.

## Modul filesystem

Das Modul **filesystem** unterstützt das Erstellen und Ändern der Größe eines Dateisystems. Dieses Modul unterstützt das Ändern der Dateisystemgröße für `ext2`, `ext3`, `ext4`, `ext4dev`, `f2fs`, `lvm`, `xfs` und `vfat`. In der folgenden Tabelle sind einige Parameter für das Modul **filesystem** aufgeführt.

Parametername	Beschreibung
dev	Name des Blockgeräts.
fstype	Dateisystemtyp.
resizefs	Erhöht die Größe des Dateisystems auf die Größe des Blockgeräts.

Das folgende Beispiel erstellt ein Dateisystem auf einer Partition.

```
- name: Create an XFS filesystem
  filesystem:
    fstype: xfs ①
    dev: /dev/vdb1 ②
```

- ① Verwendet das Dateisystem XFS.
- ② Verwendet das Gerät `/dev/vdb1`.

## Modul mount

Das Modul `mount` unterstützt die Konfiguration von Mount-Punkten auf `/etc/fstab`. In der folgenden Tabelle sind einige Parameter für das Modul `mount` aufgeführt.

Parametername	Beschreibung
<code>fstype</code>	Dateisystemtyp.
<code>opts</code>	Mount-Optionen.
<code>path</code>	Pfad des Mount-Punkts.
<code>src</code>	Gerät, das gemountet werden soll.
<code>state</code>	Mount-Status. Bei Festlegung auf <code>mounted</code> mountet das System das Gerät und konfiguriert <code>/etc/fstab</code> mit diesen Mount-Informationen. Um das Gerät zu ummounten und es aus <code>/etc/fstab</code> zu entfernen, verwenden Sie <code>absent</code> .

Im folgenden Beispiel wird ein Gerät mit einer bestimmten ID gemountet.

```
- name: Mount device with ID
  mount:
    path: /data ①
    src: UUID=a8063676-44dd-409a-b584-68be2c9f5570 ②
    fstype: xfs ③
    state: present ④
```

- ① Verwendet `/data` als Pfad für den Mount-Punkt.
- ② Mountet das Gerät mit der ID `a8063676-44dd-409a-b584-68be2c9f5570`.
- ③ Verwendet das Dateisystem XFS.
- ④ Mountet das Gerät und konfiguriert `/etc/fstab` entsprechend.

Im folgenden Beispiel wird die NFS-Freigabe unter `172.25.250.100:/share` im Verzeichnis `/nfsshare` auf dem verwalteten Host gemountet.

```
- name: Mount NFS share
  mount:
    path: /nfsshare
    src: 172.25.250.100:/share
    fstype: nfs
    opts: defaults
    dump: '0'
    passno: '0'
    state: mounted
```

## Konfigurieren des Swap mit Modulen

Red Hat Ansible Engine enthält aktuell keine Module zum Verwalten von Swap-Arbeitsspeicher. Um Swap-Arbeitsspeicher zu einem System mit Ansible mit logischen Volumes hinzuzufügen, müssen Sie eine neue Volumengruppe und ein logisches Volume mit den Modulen `lvg` und

lvol erstellen. Wenn Sie fertig sind, müssen Sie das neue logische Volume mit dem Modul command mit dem Befehl mkswap als Swap formatieren. Zuletzt müssen Sie das neue Swap-Gerät mit dem Modul command mit dem Befehl swapon aktivieren. Ansible beinhaltet die Variable ansible\_swaptotal\_mb, die den gesamten Swap-Arbeitsspeicher enthält. Mit dieser Variablen können Sie die Swap-Konfiguration und -Aktivierung auslösen, wenn der Swap-Arbeitsspeicher fast voll ist. Die folgenden Aufgaben erstellen eine Volumengruppe und ein logisches Volume für den Swap-Arbeitsspeicher, formatieren dieses logische Volume als Swap und aktivieren es.

```
- name: Create new swap VG
  lvg:
    vg: vgswap
    pvs: /dev/vda1
    state: present

- name: Create new swap LV
  lvol:
    vg: vgswap
    lv: lvswap
    size: 10g

- name: Format swap LV
  command: mkswap /dev/vgswap/lvswap
  when: ansible_swaptotal_mb < 128

- name: Activate swap LV
  command: swapon /dev/vgswap/lvswap
  when: ansible_swaptotal_mb < 128
```

## Ansible-Fakten für die Storage-Konfiguration

Ansible ruft mithilfe von Fakten Informationen über die Konfiguration der verwalteten Hosts auf den Kontrollknoten ab. Mit dem Ansible-Modul setup können Sie alle Ansible-Fakten für einen verwalteten Host abrufen.

```
[user@controlnode ~]$ ansible webservers -m setup
host.lab.example.com | SUCCESS => {
  "ansible_facts": {
    ...output omitted...
  }
}
```

Die Option filter für das Modul setup unterstützt das detaillierte Filtern basierend auf Shell-artigen Platzhaltern.

Das Element ansible\_devices enthält alle auf dem verwalteten Host verfügbaren Speichergeräte. Das Element für jedes Speichergerät enthält zusätzliche Informationen wie Partitionen oder Gesamtgröße. Das folgende Beispiel zeigt das Element ansible\_devices für einen verwalteten Host mit drei Speichergeräten: sr0, vda und vdb.

```
[user@controlnode ~]$ ansible webservers -m setup -a 'filter=ansible_devices'
host.lab.example.com | SUCCESS => {
  "ansible_facts": {
    "ansible_devices": {
      "sr0": {
        ...output omitted...
      }
    }
  }
}
```

```
        "holders": [],
        "host": "IDE interface: Intel Corporation 82371SB PIIX3 IDE
[Natoma/Triton II]",
        "links": {
            "ids": [
                "ata-QEMU_DVD-ROM_QM00003"
            ],
            "labels": [],
            "masters": [],
            "uuids": []
        },
        "model": "QEMU DVD-ROM",
        "partitions": {},
        "removable": "1",
        "rotational": "1",
        "sas_address": null,
        "sas_device_handle": null,
        "scheduler_mode": "mq-deadline",
        "sectors": "2097151",
        "sectorsize": "512",
        "size": "1024.00 MB",
        "support_discard": "0",
        "vendor": "QEMU",
        "virtual": 1
    },
    "vda": {
        "holders": [],
        "host": "SCSI storage controller: Red Hat, Inc. Virtio block
device",
        "links": {
            "ids": [],
            "labels": [],
            "masters": [],
            "uuids": []
        },
        "model": null,
        "partitions": {
            "vda1": {
                "holders": [],
                "links": {
                    "ids": [],
                    "labels": [],
                    "masters": [],
                    "uuids": [
                        "a8063676-44dd-409a-b584-68be2c9f5570"
                    ]
                },
                "sectors": "20969439",
                "sectorsize": 512,
                "size": "10.00 GB",
                "start": "2048",
                "uuid": "a8063676-44dd-409a-b584-68be2c9f5570"
            }
        },
        "removable": "0",
        "rotational": "1",
        "sas_address": null,
        "sas_device_handle": null,
        "scheduler_mode": "mq-deadline",
        "size": "10.00 GB",
        "support_discard": "0",
        "vendor": "Red Hat, Inc. Virtio"
    }
}
```

```
        "rotational": "1",
        "sas_address": null,
        "sas_device_handle": null,
        "scheduler_mode": "mq-deadline",
        "sectors": "20971520",
        "sectorsize": "512",
        "size": "10.00 GB",
        "support_discard": "0",
        "vendor": "0x1af4",
        "virtual": 1
    },
    "vdb": {
        "holders": [],
        "host": "SCSI storage controller: Red Hat, Inc. Virtio block
device",
        "links": {
            "ids": [],
            "labels": [],
            "masters": [],
            "uuids": []
        },
        "model": null,
        "partitions": {},
        "removable": "0",
        "rotational": "1",
        "sas_address": null,
        "sas_device_handle": null,
        "scheduler_mode": "mq-deadline",
        "sectors": "10485760",
        "sectorsize": "512",
        "size": "5.00 GB",
        "support_discard": "0",
        "vendor": "0x1af4",
        "virtual": 1
    }
},
"changed": false
}
```

Das Element `ansible_device_links` enthält alle verfügbaren Links für jedes Speichergerät. Das folgende Beispiel zeigt das Element `ansible_device_links` für einen verwalteten Host mit zwei Speichergeräten, `sr0` und `vda1`, die eine zugehörige ID haben.

```
[user@controlnode ~]$ ansible webservers -m setup -a 'filter=ansible_device_links'
host.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_device_links": {
            "ids": {
                "sr0": [
                    "ata-QEMU_DVD-ROM_QM00003"
                ]
            },
            "labels": {}
        }
    }
}
```

```
"masters": {},  
"uids": {  
    "vda1": [  
        "a8063676-44dd-409a-b584-68be2c9f5570"  
    ]  
},  
"changed": false  
}
```

Das Element `ansible_mounts` enthält Informationen zu den aktuell auf dem verwalteten Host gemounteten Geräten, z. B. zum gemounteten Gerät, zum Mount-Punkt und zu den Optionen. Die folgende Ausgabe zeigt das Element `ansible_mounts` für einen verwalteten Host mit einem aktiven Mount, `/dev/vda1` im Verzeichnis `/`.

```
[user@controlnode ~]$ ansible webservers -m setup -a 'filter=ansible_mounts'  
host.lab.example.com | SUCCESS => {  
    "ansible_facts": {  
        "ansible_mounts": [  
            {  
                "block_available": 2225732,  
                "block_size": 4096,  
                "block_total": 2618619,  
                "block_used": 392887,  
                "device": "/dev/vda1",  
                "fstype": "xfs",  
                "inode_available": 5196602,  
                "inode_total": 5242304,  
                "inode_used": 45702,  
                "mount": "/",  
                "options": "rw,seclabel,relatime,attr2,inode64,noquota",  
                "size_available": 9116598272,  
                "size_total": 10725863424,  
                "uuid": "a8063676-44dd-409a-b584-68be2c9f5570"  
            }  
        ],  
        "changed": false  
    }  
}
```



### Literaturhinweise

#### **parted – Configure block device partitions – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/modules/parted\\_module.html](https://docs.ansible.com/ansible/2.9/modules/parted_module.html)

#### **lvg – Configure LVM volume groups – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/modules/lvg\\_module.html](https://docs.ansible.com/ansible/2.9/modules/lvg_module.html)

#### **lvol – Configure LVM logical volumes – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/modules/lvol\\_module.html](https://docs.ansible.com/ansible/2.9/modules/lvol_module.html)

#### **filesystem – Makes a filesystem – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/modules/filesystem\\_module.html](https://docs.ansible.com/ansible/2.9/modules/filesystem_module.html)

#### **mount – Control active and configured mount points – Ansible-Dokumentation**

[https://docs.ansible.com/ansible/2.9/modules/mount\\_module.html](https://docs.ansible.com/ansible/2.9/modules/mount_module.html)

## ► Angeleitete Übung

# Verwalten von Storage

In dieser Übung partitionieren Sie ein neues Laufwerk, erstellen logische Volumes und formatieren diese mit XFS-Dateisystemen. Zudem werden Sie diese sofort und automatisch zur Bootzeit auf den verwalteten Hosts mounten.

## Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Konfigurieren von Blockgerätepartitionen mit dem Modul `parted`.
- Verwalten von LVM-Volumegruppen mit dem Modul `lvg`.
- Verwalten von logischen LVM-Volumes mit dem Modul `lvol`.
- Erstellen von Dateisystemen mit dem Modul `filesystem`.
- Steuern und Konfigurieren von Mount-Punkten in `/etc/fstab` mit dem Modul `mount`.

## Bevor Sie Beginnen

Führen Sie das Skript `lab system-storage start` über `workstation` aus, um die Umgebung für diese Übung zu konfigurieren. Das Skript erstellt das Projektverzeichnis `system-storage` und lädt die Ansible-Konfigurationsdatei und die Hostinventardatei herunter, die für die Übung erforderlich sind.

```
[student@workstation ~]$ lab system-storage start
```

## Anweisungen

Sie sind für die Verwaltung einer Reihe von Webservern zuständig. Für die Webserverkonfiguration wird empfohlen, Webserverdaten auf einer separaten Partition oder einem logischen Volume zu speichern.

Sie werden für Folgendes ein Playbook schreiben:

- Verwalten von Partitionen des Geräts `/dev/vdb`
- Verwalten einer Volumegruppe namens `apache-vg` für Webserverdaten
- Erstellen zweier logischer Volumes namens `content-lv` und `logs-lv`, die beide von der Volumegruppe `apache-vg` unterstützt werden
- Erstellen eines XFS-Dateisystems auf beiden logischen Volumes
- Mounten des logischen Volumes `content-lv` bei `/var/www`
- Mounten des logischen Volumes `logs-lv` bei `/var/log/httpd`

Wenn sich die Storage-Anforderungen für den Webserver ändern, aktualisieren Sie die entsprechenden Playbook-Variablen und führen Sie das Playbook erneut aus. Das Playbook sollte idempotent sein.

- 1. Wechseln Sie als Benutzer `student` auf `workstation` in das Arbeitsverzeichnis `/home/student/system-storage`.

```
[student@workstation ~]$ cd ~/system-storage  
[student@workstation system-storage]$
```

- 2. Überprüfen Sie die Playbook-Gerüstdatei `storage.yml` und die zugehörige Variablendatei `storage_vars.yml` im Projektverzeichnis. Führen Sie das Playbook aus.

- 2.1. Prüfen Sie das Playbook `storage.yml`.

```
---  
- name: Ensure Apache Storage Configuration  
  hosts: webservers  
  vars_files:  
    - storage_vars.yml  
  tasks:  
    - name: Correct partitions exist on /dev/vdb  
      debug:  
        msg: TODO  
      loop: "{{ partitions }}"  
  
    - name: Ensure Volume Groups Exist  
      debug:  
        msg: TODO  
      loop: "{{ volume_groups }}"  
  
    - name: Create each Logical Volume (LV) if needed  
      debug:  
        msg: TODO  
      loop: "{{ logical_volumes }}"  
      when: true  
  
    - name: Ensure XFS Filesystem exists on each LV  
      debug:  
        msg: TODO  
      loop: "{{ logical_volumes }}"  
  
    - name: Ensure the correct capacity for each LV  
      debug:  
        msg: TODO  
      loop: "{{ logical_volumes }}"  
  
    - name: Each Logical Volume is mounted  
      debug:  
        msg: TODO  
      loop: "{{ logical_volumes }}"
```

Der Name der einzelnen Aufgaben dient als Übersicht über das beabsichtigte Implementierungsverfahren. In späteren Schritten werden Sie diese sechs Aufgaben aktualisieren und ändern.

- 2.2. Überprüfen Sie die Variablendatei `storage_vars.yml`.

```
---
```

```
partitions:
  - number: 1
    start: 1MiB
    end: 257MiB

volume_groups:
  - name: apache-vg
    devices: /dev/vdb1

logical_volumes:
  - name: content-lv
    size: 64M
    vgroup: apache-vg
    mount_path: /var/www

  - name: logs-lv
    size: 128M
    vgroup: apache-vg
    mount_path: /var/log/httpd
```

Diese Datei beschreibt die vorgesehene Struktur von Partitionen, Volumegruppen und logischen Volumes auf den einzelnen Webservern. Die erste Partition beginnt bei einem Offset von 1 MiB vom Anfang des Geräts /dev/vdb und endet bei einem Offset von 257 MiB bei einer Gesamtgröße von 256 MiB.

Jeder Webserver verfügt über eine Volumegruppe namens apache-vg, die die erste Partition des Geräts /dev/vdb enthält.

Jeder Webserver verfügt über zwei logische Volumes. Das erste logische Volume erhält die Bezeichnung content-lv und eine Größe von 64 MiB und wird an die Volumegruppe apache-vg angehängt sowie bei /var/www gemountet. Das zweite logische Volume erhält die Bezeichnung logs-lv und eine Größe von 128 MiB und wird an die Volumegruppe apache-vg angehängt sowie bei /var/log/httpd gemountet.



### Anmerkung

Die Volumegruppe apache-vg besitzt eine Kapazität von 256 MiB, da sie von der Partition /dev/vdb1 unterstützt wird. Sie bietet ausreichend Kapazität für beide logischen Volumes.

2.3. Führen Sie das Playbook storage.yml aus.

```
[student@workstation system-storage]$ ansible-playbook storage.yml

PLAY [Ensure Apache Storage Configuration] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Correct partitions exist on /dev/vdb] ****
```

```

ok: [servera.lab.example.com] => (item={u'start': u'1MiB', u'end': u'257MiB',
u'number': 1}) => {
    "msg": "TODO"
}

...output omitted...

TASK [Each Logical Volume is mounted] ****
ok: [servera.lab.example.com] => (item={u'vgroup': u'apache-vg', u'size': u'64M',
u'mount_path': u'/var/www', u'name': u'content-lv'}) => {
    "msg": "TODO"
}
ok: [servera.lab.example.com] => (item={u'vgroup': u'apache-vg', u'size': u'128M',
u'mount_path': u'/var/log/httpd', u'name': u'logs-lv'}) => {
    "msg": "TODO"
}

PLAY RECAP ****
servera.lab.example.com      : ok=7      changed=0      unreachable=0      failed=0

```

- 3. Ändern Sie die erste Aufgabe, damit sie das Modul `parted` zum Konfigurieren einer Partition für die einzelnen Loopelemente verwendet. Jedes Element beschreibt eine geplante Partition des Geräts `/dev/vdb` auf den einzelnen Webservern:

#### **number**

Die Partitionsnummer. Verwenden Sie dieses Element als Wert des Keywords `number` für das Modul `parted`.

#### **start**

Der Anfang der Partition als Offset vom Anfang des Blockgeräts. Verwenden Sie dieses Element als Wert des Keywords `part_start` für das Modul `parted`.

#### **end**

Das Ende der Partition als Offset vom Anfang des Blockgeräts. Verwenden Sie dieses Element als Wert des Keywords `part_end` für das Modul `parted`.

Die erste Aufgabe sollte folgenden Inhalt aufweisen:

```

- name: Correct partitions exist on /dev/vdb
  parted:
    device: /dev/vdb
    state: present
    number: "{{ item.number }}"
    part_start: "{{ item.start }}"
    part_end: "{{ item.end }}"
    loop: "{{ partitions }}"

```

- 4. Ändern Sie die zweite Aufgabe des Plays, um das Modul `lvgl` zum Konfigurieren einer Volumegruppe für die einzelnen Loopelemente zu verwenden. Jedes Element der

Variablen `volume_groups` beschreibt eine Volumegruppe, die auf jedem Webserver vorhanden sein sollte:

**name**

Der Name der Volumegruppe. Verwenden Sie dieses Element als Wert des Keywords `vg` für das Modul `lvg`.

**devices**

Eine durch Komma getrennte Liste von Geräten oder Partitionen, die die Volumegruppe bilden. Verwenden Sie dieses Element als Wert des Keywords `pvs` für das Modul `lvg`.

Die zweite Aufgabe sollte folgenden Inhalt aufweisen:

```
- name: Ensure Volume Groups Exist
  lvg:
    vg: "{{ item.name }}"
    pvs: "{{ item.devices }}"
  loop: "{{ volume_groups }}
```

- 5. Ändern Sie die dritte Aufgabe des Plays, um das Modul `lvol` zum Erstellen eines logischen Volumes für die einzelnen Elemente zu verwenden. Verwenden Sie die Keywords des Elements, um das neue logische Volume zu erstellen:

**name**

Der Name des logischen Volumes. Verwenden Sie dieses Element als Wert des Keywords `lv` für das Modul `lvol`.

**vgroup**

Der Name der Volumegruppe, die Storage für das logische Volume bereitstellt.

**size**

Die Größe des logischen Volumes. Der Wert dieses Keywords ist ein beliebiger zulässiger Wert für die `-L`-Option des Befehls `lvcreate`.

Führen Sie die Aufgabe nur aus, wenn noch kein logisches Volume vorhanden ist. Aktualisieren Sie die Anweisung `when`, um zu überprüfen, dass kein logisches Volume mit einem Namen vorhanden ist, der dem Keyword `name` des Elements entspricht.

- 5.1. Ändern Sie die dritte Aufgabe, damit diese das Modul `lvol` verwendet. Legen Sie den Namen der Volumegruppe sowie den Namen und die Größe des logischen Volumes mithilfe der Keywords der einzelnen Elemente fest. Die dritte Aufgabe umfasst jetzt den folgenden Inhalt:

```
- name: Create each Logical Volume (LV) if needed
  lvol:
    vg: "{{ item.vgroup }}"
    lv: "{{ item.name }}"
    size: "{{ item.size }}"
  loop: "{{ logical_volumes }}"
  when: true
```

- 5.2. Der Ansible-Fakt `ansible_lvm` enthält Informationen zu Objekten für die Verwaltung logischer Volumes auf den einzelnen Hosts. Verwenden Sie einen

Ad-hoc-Befehl, um den aktuellen Satz logischer Volumes auf dem Remote-Host anzuzeigen:

```
[student@workstation system-storage]$ ansible all -m setup -a \
> "filter=ansible_lvm"
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_lvm": {
            "lvs": {},
            "pvs": {},
            "vgs": {}
        },
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false
}
```

Der Wert des Keywords `lvs` gibt an, dass sich auf dem Remote-Host keine logischen Volumes befinden.

- 5.3. Führen Sie das Playbook aus, um die logischen Volumes auf dem Remote-Host zu erstellen.

```
[student@workstation system-storage]$ ansible-playbook storage.yml

PLAY [Ensure Apache Storage Configuration] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Correct partitions exist on /dev/vdb] ****
changed: [servera.lab.example.com] => (item={...output omitted...})

TASK [Ensure Volume Groups Exist] ****
changed: [servera.lab.example.com] => (item={...output omitted...})

TASK [Create each Logical Volume (LV) if needed] ****
changed: [servera.lab.example.com] => (item={...output omitted...})
changed: [servera.lab.example.com] => (item={...output omitted...})

TASK [Ensure XFS Filesystem exists on each LV] ****
ok: [servera.lab.example.com] => (item={...output omitted...}) => {
    "msg": "TODO"
}
...output omitted...
PLAY RECAP ****
servera.lab.example.com      : ok=7      changed=3      unreachable=0      failed=0
```

- 5.4. Führen Sie einen weiteren Ad-hoc-Befehl aus, um die Struktur der Variablen `ansible_lvm` anzuzeigen, wenn logische Volumes auf dem Remote-Host vorhanden sind.

```
[student@workstation system-storage]$ ansible all -m setup -a \
> "filter=ansible_lvm"
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_lvm": {
            "lvs": {❶
                "content-lv": {
                    "size_g": "0.06",
                    "vg": "apache-vg"
                },
                "logs-lv": {
                    "size_g": "0.12",
                    "vg": "apache-vg"
                }
            },
            "pvs": {❷
                "/dev/vdb1": {
                    "free_g": "0.06",
                    "size_g": "0.25",
                    "vg": "apache-vg"
                }
            },
            "vgs": {❸
                "apache-vg": {
                    "free_g": "0.06",
                    "num_lvs": "2",
                    "num_pvs": "1",
                    "size_g": "0.25"
                }
            }
        },
        "changed": false
    }
}
```

- ❶ Der Wert des Keywords `lvs` ist die Datenstruktur eines Schlüssel-/Wert-Paars. Die Schlüssel dieser Struktur sind die Namen von beliebigen logischen Volumes auf dem Host. Dies zeigt an, dass die logischen Volumes `content-lv` und `logs-lv` vorhanden sind. Für jedes logische Volume wird die entsprechende Volumegruppe durch das Keyword `vg` bereitgestellt.
  - ❷ Das Keyword `pvs` enthält Informationen zu physischen Volumes auf dem Host. Die Informationen zeigen an, dass die Partition `/dev/vdb1` zur Volumegruppe `apache-vg` gehört.
  - ❸ Das Keyword `vgs` enthält Informationen zu Volumegruppen auf dem Host.
- 5.5. Aktualisieren Sie die Anweisung `when`, um zu überprüfen, dass kein logisches Volume mit einem Namen vorhanden ist, der dem Keyword `name` des Elements entspricht. Die dritte Aufgabe umfasst jetzt den folgenden Inhalt:

```
- name: Create each Logical Volume (LV) if needed
  lvol:
    vg: "{{ item.vgroup }}"
    lv: "{{ item.name }}"
    size: "{{ item.size }}"
  loop: "{{ logical_volumes }}"
  when: item.name not in ansible_lvm["lvs"]
```

- 6. Ändern Sie die vierte Aufgabe, um das Modul `filesystem` zu verwenden. Konfigurieren Sie die Aufgabe, um sicherzustellen, dass jedes logische Volume als XFS-Dateisystem formatiert ist. Denken Sie daran, dass ein logisches Volume dem logischen Gerät /dev/<Name der Volume-Gruppe>/<Name des logischen Volumes> zugeordnet ist.

Die vierte Aufgabe sollte folgenden Inhalt aufweisen:

```
- name: Ensure XFS Filesystem exists on each LV
  filesystem:
    dev: "/dev/{{ item.vgroup }}/{{ item.name }}"
    fstype: xfs
  loop: "{{ logical_volumes }}"
```

- 7. Konfigurieren Sie die fünfte Aufgabe, um sicherzustellen, dass jedes logische Volume über die richtige Storage-Kapazität verfügt. Wenn die Kapazität des logischen Volumes zunimmt, muss das Dateisystem des Volumes erweitert werden.



### Warnung

Wenn die Kapazität eines logischen Volumes verringert werden muss, tritt bei dieser Aufgabe ein Fehler auf, da eine abnehmende Kapazität vom XFS-Dateisystem nicht unterstützt wird.

Die fünfte Aufgabe sollte folgenden Inhalt aufweisen:

```
- name: Ensure the correct capacity for each LV
  lvol:
    vg: "{{ item.vgroup }}"
    lv: "{{ item.name }}"
    size: "{{ item.size }}"
    resizefs: yes
    force: yes
  loop: "{{ logical_volumes }}"
```

- 8. Stellen Sie mithilfe des Moduls `mount` in der sechsten Aufgabe sicher, dass jedes logische Volume im entsprechenden Mount-Pfad bereitgestellt wird und nach einem Reboot bestehen bleibt.

Die sechste Aufgabe sollte folgenden Inhalt aufweisen:

```
- name: Each Logical Volume is mounted
  mount:
    path: "{{ item.mount_path }}"
    src: "/dev/{{ item.vgroup }}/{{ item.name }}"
    fstype: xfs
    opts: noatime
    state: mounted
  loop: "{{ logical_volumes }}"
```

- 9. Prüfen Sie das abgeschlossene Playbook `storage.yml`. Führen Sie das Playbook aus und verifizieren Sie, dass alle logischen Volumes gemountet sind.

9.1. Prüfen Sie das Playbook:

```
---
- name: Ensure Apache Storage Configuration
  hosts: webservers
  vars_files:
    - storage_vars.yml
  tasks:
    - name: Correct partitions exist on /dev/vdb
      parted:
        device: /dev/vdb
        state: present
        number: "{{ item.number }}"
        part_start: "{{ item.start }}"
        part_end: "{{ item.end }}"
      loop: "{{ partitions }}"
    - name: Ensure Volume Groups Exist
      lvg:
        vg: "{{ item.name }}"
        pvs: "{{ item.devices }}"
      loop: "{{ volume_groups }}"
    - name: Create each Logical Volume (LV) if needed
      lvlv:
        vg: "{{ item.vgroup }}"
        lv: "{{ item.name }}"
        size: "{{ item.size }}"
      loop: "{{ logical_volumes }}"
      when: item.name not in ansible_lvm["lvs"]
    - name: Ensure XFS Filesystem exists on each LV
      filesystem:
        dev: "/dev/{{ item.vgroup }}/{{ item.name }}"
        fstype: xfs
      loop: "{{ logical_volumes }}"
```

```
- name: Ensure the correct capacity for each LV
  lvol:
    vg: "{{ item.vgroup }}"
    lv: "{{ item.name }}"
    size: "{{ item.size }}"
    resizefs: yes
    force: yes
  loop: "{{ logical_volumes }}"

- name: Each Logical Volume is mounted
  mount:
    path: "{{ item.mount_path }}"
    src: "/dev/{{ item.vgroup }}/{{ item.name }}"
    fstype: xfs
    opts: noatime
    state: mounted
  loop: "{{ logical_volumes }}"
```

## 9.2. Führen Sie das Playbook aus.

```
[student@workstation system-storage]$ ansible-playbook storage.yml

PLAY [Ensure Apache Storage Configuration] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Correct partitions exist on /dev/vdb] ****
ok: [servera.lab.example.com] => (item={...output omitted...})

TASK [Ensure Volume Groups Exist] ****
ok: [servera.lab.example.com] => (item={...output omitted...})
...output omitted...

TASK [Create each Logical Volume (LV) if needed] ****
skipping: [servera.lab.example.com] => (item={...output omitted...})
skipping: [servera.lab.example.com] => (item={...output omitted...})

TASK [Ensure XFS Filesystem exists on each LV] ****
changed: [servera.lab.example.com] => (item={...output omitted...})
changed: [servera.lab.example.com] => (item={...output omitted...})

TASK [Ensure the correct capacity for each LV] ****
ok: [servera.lab.example.com] => (item={...output omitted...})
ok: [servera.lab.example.com] => (item={...output omitted...})

TASK [Each Logical Volume is mounted] ****
changed: [servera.lab.example.com] => (item={...output omitted...})
changed: [servera.lab.example.com] => (item={...output omitted...})

PLAY RECAP ****
servera.lab.example.com      : ok=6      changed=2      unreachable=0      failed=0
skipped=1      rescued=0      ignored=0
```

Eine Aufgabe wird während der Ausführung übersprungen, da das Playbook zuvor mit denselben Variablenwerten ausgeführt wurde. Die logischen Volumes mussten nicht erstellt werden.

- 9.3. Verwenden Sie einen Ad-hoc-Befehl von Ansible, um den Befehl `lsblk` auf dem Remote-Host auszuführen. Die Ausgabe gibt die Mount-Punkte für die logischen Volumes an.

```
[student@workstation system-storage]$ ansible all -a lsblk
servera.lab.example.com | CHANGED | rc=0 >>
NAME          MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0           11:0    1 1024M  0 rom
vda           252:0   0  10G  0 disk
└─vda1        252:1   0  10G  0 part /
vdb           252:16  0   1G  0 disk
└─vdb1        252:17  0  256M 0 part
  ├─apache--vg-content--lv 253:0   0   64M 0 lvm  /var/www
  └─apache--vg-logs--lv   253:1   0  128M 0 lvm  /var/log/httpd
```

- 10. Erhöhen Sie die Kapazität des logischen Volumes `content-lv` auf 128 MiB und die Kapazität von `logs-lv` auf 256 MiB. Dies erfordert eine Erhöhung der Kapazität der Volumegruppe `apache-vg`.

Erstellen Sie eine neue Partition mit einer Kapazität von 256 MiB und fügen Sie sie der Volumegruppe `apache-vg` hinzu.

- 10.1. Bearbeiten Sie die Variablendefinition `partitions` in der Datei `storage_vars.yml`, um dem Gerät `/dev/vdb` eine zweite Partition hinzuzufügen. Die Variable `partitions` sollte folgenden Inhalt aufweisen:

```
partitions:
  - number: 1
    start: 1MiB
    end: 257MiB
  - number: 2
    start: 257MiB
    end: 513MiB
```

- 10.2. Bearbeiten Sie die Variablendefinition `volume_groups` in der Datei `storage_vars.yml`. Fügen Sie die zweite Partition der Liste der Geräte hinzu, die die Volumegruppe unterstützen. Die Variable `volume_groups` sollte folgenden Inhalt aufweisen:

```
volume_groups:
  - name: apache-vg
    devices: /dev/vdb1, /dev/vdb2
```

- 10.3. Verdoppeln Sie die Kapazität der einzelnen logischen Volumes, die in der Datei `storage_vars.yml` definiert sind. Die Variable `logical_volumes` sollte folgenden Inhalt aufweisen:

```
logical_volumes:
  - name: content-lv
    size: 128M
    vgroup: apache-vg
    mount_path: /var/www

  - name: logs-lv
    size: 256M
    vgroup: apache-vg
    mount_path: /var/log/httpd
```

10.4. Führen Sie das Playbook aus. Verifizieren Sie die neue Kapazität der einzelnen logischen Volumes.

```
[student@workstation system-storage]$ ansible-playbook storage.yml

PLAY [Ensure Apache Storage Configuration] ****

TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Correct partitions exist on /dev/vdb] ****
ok: [servera.lab.example.com] => (item={...output omitted...})
changed: [servera.lab.example.com] => (item={u'start': u'257MiB', u'end': u'513MiB', u'number': 2})

TASK [Ensure Volume Groups Exist] ****
changed: [servera.lab.example.com] => (item={u'name': u'apache-vg', u'devices': u'/dev/vdb1,/dev/vdb2'})
...output omitted...

TASK [Create each Logical Volume (LV) if needed] ****
skipping: [servera.lab.example.com] => (item={u'vgroup': u'apache-vg', u'size': u'128M', u'mount_path': u'/var/www', u'name': u'content-lv'})
skipping: [servera.lab.example.com] => (item={u'vgroup': u'apache-vg', u'size': u'256M', u'mount_path': u'/var/log/httpd', u'name': u'logs-lv'})

TASK [Ensure XFS Filesystem exists on each LV] ****
ok: [servera.lab.example.com] => (item={...output omitted...})
ok: [servera.lab.example.com] => (item={...output omitted...})

TASK [Ensure the correct capacity for each LV] ****
changed: [servera.lab.example.com] => (item={u'vgroup': u'apache-vg', u'size': u'128M', u'mount_path': u'/var/www', u'name': u'content-lv'})
changed: [servera.lab.example.com] => (item={u'vgroup': u'apache-vg', u'size': u'256M', u'mount_path': u'/var/log/httpd', u'name': u'logs-lv'})

TASK [Each Logical Volume is mounted] ****
ok: [servera.lab.example.com] => (item={...output omitted...})
ok: [servera.lab.example.com] => (item={...output omitted...})
```

## Kapitel 9 | Automatisieren von Linux-Administrationsaufgaben

```
PLAY RECAP ****
servera.lab.example.com : ok=6    changed=3    unreachable=0    failed=0
skipped=1    rescued=0    ignored=0
```

Die Ausgabe zeigt Änderungen an den Partitionen und an der Volumegruppe auf dem Remote-Host sowie an der Größe der beiden logischen Volumes an.

- 10.5. Verwenden Sie einen Ad-hoc-Befehl von Ansible, um den Befehl `lsblk` auf dem Remote-Host auszuführen.

```
[student@workstation system-storage]$ ansible all -a lsblk
servera.lab.example.com | CHANGED | rc=0 >>
NAME          MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0           11:0    1 1024M  0 rom
vda           252:0   0  10G  0 disk
└─vda1        252:1   0  10G  0 part /
vdb           252:16  0   1G  0 disk
└─vdb1        252:17  0  256M 0 part
  ├─apache--vg-content--lv 253:0   0 128M 0 lvm  /var/www
  └─apache--vg-logs--lv   253:1   0 256M 0 lvm  /var/log/httpd
└─vdb2        252:18  0  256M 0 part
  ├─apache--vg-content--lv 253:0   0 128M 0 lvm  /var/www
  └─apache--vg-logs--lv   253:1   0 256M 0 lvm  /var/log/httpd
```

Die Ausgabe zeigt an, dass jedes logische Volume die richtige Größe aufweist und im richtigen Verzeichnis gemountet wurde. Für jedes logische Volume sind zwei Einträge vorhanden, da sich auf dem logischen Volume gespeicherte Dateien möglicherweise physisch auf einer der Partitionen (`/dev/vdb1` oder `/dev/vdb2`) befinden.

## Beenden

Führen Sie den Befehl `lab system-storage finish` aus, um den verwalteten Host zu bereinigen.

```
[student@workstation ~]$ lab system-storage finish
```

Hiermit ist die angeleitete Übung beendet.

# Verwalten der Netzwerkkonfiguration

## Ziele

Am Ende dieses Abschnitts sollten Sie zu Folgendem in der Lage sein: Netzwerkeinstellungen und Namensauflösung auf verwalteten Hosts konfigurieren und networkbezogene Ansible-Fakten sammeln.

## Konfigurieren des Netzwerks mit der Netzwerksystemrolle

Red Hat Enterprise Linux 8 enthält eine Sammlung von Ansible-Systemrollen, um RHEL-basierte Systeme zu konfigurieren. Das Paket *rhel-system-roles* installiert diese Systemrollen, die beispielsweise die Konfiguration der Zeitsynchronisation oder des Netzwerks unterstützen. Mit dem Befehl `ansible-galaxy list` können Sie die aktuell installierten Systemrollen auflisten.

```
[user@controlnode ~]$ ansible-galaxy list
- linux-system-roles.kdump, (unknown version)
- linux-system-roles.network, (unknown version)
- linux-system-roles.postfix, (unknown version)
- linux-system-roles.selinux, (unknown version)
- linux-system-roles.timesync, (unknown version)
- rhel-system-roles.kdump, (unknown version)
- rhel-system-roles.network, (unknown version)
- rhel-system-roles.postfix, (unknown version)
- rhel-system-roles.selinux, (unknown version)
- rhel-system-roles.timesync, (unknown version)
```

Die Rollen befinden sich im Verzeichnis `/usr/share/ansible/roles`. Jede Rolle, die mit `linux-system-roles` beginnt, ist ein Symlink zur entsprechenden `rhel-system-roles`-Rolle.

Die Netzwerksystemrolle unterstützt die Konfiguration des Netzwerks auf verwalteten Hosts. Diese Rolle unterstützt die Konfiguration von Ethernet-Schnittstellen, Bridge-Schnittstellen, verbundenen Schnittstellen, VLAN-Schnittstellen, MacVLAN-Schnittstellen und Infiniband-Schnittstellen. Die Netzwerkrolle ist mit zwei Variablen konfiguriert: `network_provider` und `network_connections`.

```
---
network_provider: nm
network_connections:
  - name: ens4
    type: ethernet
    ip:
      address:
        - 172.25.250.30/24
```

Die Variable `network_provider` konfiguriert den Back-End-Anbieter, `nm` (NetworkManager) oder `initscripts`. Unter Red Hat Enterprise Linux 8 verwendet die Netzwerkrolle `nm`

(NetworkManager) als Standard-Netzwerkanbieter. Der `initscripts`-Anbieter wird für RHEL 6-Systeme verwendet und erfordert, dass der Service `network` verfügbar ist. Die Variable `network_connections` konfiguriert die verschiedenen Verbindungen, die als eine Liste von Wörterbüchern angegeben sind, wobei der Schnittstellenname als Verbindungsname verwendet wird.

In der folgenden Tabelle sind die Optionen für die Variable `network_connections` aufgelistet.

Optionsname	Beschreibung
<code>name</code>	Identifiziert das Verbindungsprofil.
<code>state</code>	Der Laufzeitstatus eines Verbindungsprofils. Entweder <code>up</code> , wenn das Verbindungsprofil aktiv ist, oder <code>down</code> , wenn es nicht aktiv ist.
<code>persistent_state</code>	Identifiziert, ob ein Verbindungsprofil persistent ist. Entweder <code>present</code> , wenn das Verbindungsprofil persistent ist, oder <code>absent</code> , wenn es nicht persistent ist.
<code>type</code>	Identifiziert den Verbindungstyp. Gültige Werte sind <code>ethernet</code> , <code>bridge</code> , <code>bond</code> , <code>team</code> , <code>vlan</code> , <code>macvlan</code> und <code>infiniband</code> .
<code>autoconnect</code>	Legt fest, ob die Verbindung automatisch gestartet wird.
<code>mac</code>	Beschränkt die Verbindung auf Geräte mit einer bestimmten MAC-Adresse.
<code>interface_name</code>	Beschränkt das Verbindungsprofil auf eine bestimmte Schnittstelle.
<code>zone</code>	Konfiguriert die FirewallD-Zone für die Schnittstelle.
<code>ip</code>	Legt die IP-Konfiguration für die Verbindung fest. Unterstützt Optionen wie zum Beispiel <code>address</code> , um eine statische IP-Adresse anzugeben, oder <code>dns</code> , um einen DNS-Server zu konfigurieren.

Im folgenden Beispiel werden einige der vorherigen Optionen verwendet:

```
network_connections:
- name: eth0 ①
  persistent_state: present ②
  type: ethernet ③
  autoconnect: yes ④
  mac: 00:00:5e:00:53:5d ⑤
  ip:
    address:
      - 172.25.250.40/24 ⑥
  zone: external ⑦
```

- ❶ Verwendet `eth0` als Verbindungsnamen.
- ❷ Macht die Verbindung persistent. Dies ist der Standardwert.
- ❸ Legt den Verbindungstyp auf `ethernet` fest.
- ❹ Startet die Verbindung beim Bootvorgang automatisch. Dies ist der Standardwert.
- ❺ Beschränkt die Verbindungsnutzung auf ein Gerät mit dieser MAC-Adresse.
- ❻ Konfiguriert die IP-Adresse `172.25.250.40/24` für die Verbindung.
- ❼ Konfiguriert die externe Zone als FirewallD-Zone der Verbindung.

Um die Netzwerksystemrolle zu verwenden, müssen Sie den Rollennamen unter der Klausel `roles` in Ihrem Playbook wie folgt angeben:

```
- name: NIC Configuration
  hosts: webservers
  vars:
    network_connections:
      - name: ens4
        type: ethernet
        ip:
          address:
            - 172.25.250.30/24
  roles:
    - rhel-system-roles.network
```

Sie können Variablen für die Netzwerkrolle mit der Klausel `vars` angeben (wie im vorherigen Beispiel) oder eine YAML-Datei mit diesen Variablen unter den Verzeichnissen `group_vars` oder `host_vars` erstellen, je nach Anwendungsfall.

## Konfigurieren des Netzwerks mit Modulen

Als Alternative zur Systemrolle `network` enthält Ansible eine Sammlung von Modulen, die die Netzwerkkonfiguration auf einem System unterstützen. Das Modul `nmcli` unterstützt die Verwaltung von Netzwerkverbindungen und Geräten. Dieses Modul unterstützt die Konfiguration von Teaming und Bonding für Netzwerkschnittstellen sowie die IPv4- und IPv6-Adressierung.

In der folgenden Tabelle sind einige Parameter für das Modul `nmcli` aufgeführt.

Parametername	Beschreibung
<code>conn_name</code>	Konfiguriert den Verbindungsnamen.
<code>autoconnect</code>	Ermöglicht die automatische Verbindungsaktivierung beim Booten.
<code>dns4</code>	Konfiguriert DNS-Server für IPv4 (bis zu 3).
<code>gw4</code>	Konfiguriert das IPv4-Gateway für die Schnittstelle.
<code>ifname</code>	Schnittstelle, die an die Verbindung gebunden werden soll.

Parametername	Beschreibung
ip4	IP-Adresse (IPv4) für die Schnittstelle.
state	Aktiviert oder deaktiviert die Netzwerkschnittstelle.
type	Typ des Geräts oder der Netzwerkverbindung.

Im folgenden Beispiel wird eine statische IP-Konfiguration für eine Netzwerkverbindung und ein Gerät konfiguriert.

```
- name: NIC configuration
  nmcli:
    conn_name: ens4-conn ❶
    ifname: ens4 ❷
    type: ethernet ❸
    ip4: 172.25.250.30/24 ❹
    gw4: 172.25.250.1 ❺
    state: present ❻
```

- ❶ Konfiguriert ens4-conn als Verbindungsnamen.
- ❷ Bindet die ens4-conn-Verbindung an die ens4-Netzwerkschnittstelle.
- ❸ Konfiguriert die Netzwerkschnittstelle als ethernet.
- ❹ Konfiguriert die IP-Adresse 172.25.250.30/24 an der Schnittstelle.
- ❺ Legt das Gateway auf 172.25.250.1 fest.
- ❻ Stellt sicher, dass die Verbindung verfügbar ist.

Das Modul `hostname` legt den Hostnamen für einen verwalteten Host fest, ohne die Datei /etc/hosts zu ändern. Dieses Modul verwendet den Parameter `name` zum Angeben des neuen Hostnamens, wie in der unten gezeigten Aufgabe:

```
- name: Change hostname
  hostname:
    name: managedhost1
```

Das Modul `firewalld` unterstützt die Verwaltung von FirewallD auf verwalteten Hosts. Dieses Modul unterstützt die Konfiguration von FirewallD-Regeln für Services und Ports. Es unterstützt auch die Zonenverwaltung, einschließlich der Zuordnung von Netzwerkschnittstellen und Regeln zu einer bestimmten Zone.

Die folgende Aufgabe zeigt, wie Sie eine FirewallD-Regel für den Service `http` in der Standardzone (`public`) erstellen. Die Aufgabe konfiguriert die Regel als permanent und stellt sicher, dass sie aktiv ist.

```
- name: Enabling http rule
firewalld:
  service: http
  permanent: yes
  state: enabled
```

Diese Aufgabe konfiguriert eth0 in der externen FirewallD-Zone.

```
- name: Moving eth0 to external
firewalld:
  zone: external
  interface: eth0
  permanent: yes
  state: enabled
```

In der folgenden Tabelle sind einige Parameter für das Modul `firewalld` aufgeführt.

Parametername	Beschreibung
interface	Name der Schnittstelle, die mit FirewallD verwaltet werden soll.
port	Port oder Portbereich. Verwendet das Format Port/Protokoll oder Port-Port/Protokoll.
rich_rule	Umfangreiche Regel für FirewallD.
service	Name des Service, der mit FirewallD verwaltet werden soll.
source	Quellnetzwerk, das mit FirewallD verwaltet werden soll.
zone	FirewallD-Zone.
state	Aktiviert oder deaktiviert eine FirewallD-Konfiguration.
type	Typ des Geräts oder der Netzwerkverbindung.

## Ansible-Fakten für die Netzwerkkonfiguration

Ansible ruft mithilfe von Fakten Informationen über die Konfiguration der verwalteten Hosts auf den Kontrollknoten ab. Mit dem Ansible-Modul `setup` können Sie alle Ansible-Fakten für einen verwalteten Host abrufen.

```
[user@controlnode ~]$ ansible webserver -m setup
host.lab.example.com | SUCCESS => {
    "ansible_facts": {
        ...output omitted...
    }
}
```

Alle Netzwerkschnittstellen für einen verwalteten Host sind unter dem Element `ansible_interfaces` verfügbar. Mit dem Parameter `gather_subset=network` für das Modul `setup` können Sie die Fakten auf die Fakten beschränken, die in der Teilmenge `network` enthalten sind. Die Option `filter` für das Modul `setup` unterstützt das detaillierte Filtern basierend auf Shell-artigen Platzhaltern.

```
[user@controlnode ~]$ ansible webservers -m setup \
> -a 'gather_subset=network filter=ansible_interfaces'
host.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_interfaces": [
            "ens4",
            "lo",
            "ens3"
        ],
        "changed": false
    }
}
```

Der vorherige Befehl zeigt, dass drei Netzwerkschnittstellen auf dem verwalteten Host `host.lab.example.com` verfügbar sind: `lo`, `ens3` und `ens4`.

Mit dem Filter `ansible_NIC_name` für das Modul `setup` können Sie zusätzliche Informationen über die Konfiguration für eine Netzwerkschnittstelle abrufen. Um beispielsweise die Konfiguration für die Netzwerkschnittstelle `ens4` abzurufen, verwenden Sie den Filter `ansible_ens4`.

```
[user@controlnode ~]$ ansible webservers -m setup \
> -a 'gather_subset=network filter=ansible_ens4'
host.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_ens4": {
            "active": true,
            "device": "ens4",
            "features": {},
            "hw_timestamp_filters": [],
            "ipv4": {
                "address": "172.25.250.30",
                "broadcast": "172.25.250.255",
                "netmask": "255.255.255.0",
                "network": "172.25.250.0"
            },
            "ipv6": [
                {
                    "address": "fe80::5b42:8c94:1fc7:40ae",
                    "prefix": "64",
                    "scope": "link"
                }
            ],
            "macaddress": "52:54:00:01:fa:0a",
            "module": "virtio_net",
            "mtu": 1500,
            "pciid": "virtio1",
            "promisc": false,
            "state": "up"
        }
    }
}
```

```
        "speed": -1,
        "timestamping": [
            "tx_software",
            "rx_software",
            "software"
        ],
        "type": "ether"
    },
    "changed": false
}
```

Der vorherige Befehl zeigt zusätzliche Konfigurationsdetails wie die IP-Adressenkonfiguration für IPv4 und IPv6, das zugeordnete Gerät und den Typ an.

In der folgenden Tabelle sind einige der verfügbaren Fakten für die Teilmenge `network` aufgeführt.

Faktenname	Beschreibung
ansible_dns	Beinhaltet die IP-Adresse der DNS-Server und die Suchdomäne(n).
ansible_domain	Enthält die Unterdomäne für den verwalteten Host.
ansible_all_ipv4_addresses	Enthält alle IPv4-Adressen, die auf dem verwalteten Host konfiguriert sind.
ansible_all_ipv6_addresses	Enthält alle IPv6-Adressen, die auf dem verwalteten Host konfiguriert sind.
ansible_fqdn	Enthält den FQDN für den verwalteten Host.
ansible_hostname	Enthält den nicht qualifizierten Hostnamen, die Zeichenfolge im FQDN vor dem ersten Punkt.
ansible_nodename	Enthält den vom System gemeldeten Hostnamen für den verwalteten Host.



### Anmerkung

Ansible stellt auch die Variable `inventory_hostname` bereit, die den in der Inventardatei von Ansible konfigurierten Hostnamen enthält.



## Literaturhinweise

### **Knowledgebase: Red Hat Enterprise Linux (RHEL) System Roles**

<https://access.redhat.com/articles/3050101>

### **Linux System Roles**

<https://linux-system-roles.github.io/>

### **nmcli Module Documentation**

[https://docs.ansible.com/ansible/2.9/modules/nmcli\\_module.html](https://docs.ansible.com/ansible/2.9/modules/nmcli_module.html)

### **hostname Module Documentation**

[https://docs.ansible.com/ansible/2.9/modules/hostname\\_module.html](https://docs.ansible.com/ansible/2.9/modules/hostname_module.html)

### **firewalld Module Documentation**

[https://docs.ansible.com/ansible/2.9/modules/firewalld\\_module.html](https://docs.ansible.com/ansible/2.9/modules/firewalld_module.html)

## ► Angeleitete Übung

# Verwalten der Netzwerkkonfiguration

In dieser Übung passen Sie die Netzwerkkonfiguration eines verwalteten Hosts an und sammeln Informationen dazu in einer von einer Vorlage erstellten Datei.

## Ergebnisse

Sie sollten in der Lage sein, Netzwerkeinstellungen und Namensauflösung für verwaltete Hosts zu konfigurieren und netzwerkbezogene Ansible-Fakten zu sammeln.

## Bevor Sie Beginnen

Führen Sie das Skript `lab system-network start` über `workstation` aus, um die Umgebung für diese Übung zu konfigurieren. Das Skript erstellt das Arbeitsverzeichnis `system-network` und lädt die Ansible-Konfigurationsdatei und die Hostinventardatei herunter, die für die Übung erforderlich sind.

```
[student@workstation ~]$ lab system-network start
```

## Anweisungen

- 1. Überprüfen Sie die Inventardatei im Verzeichnis `/home/student/system-network`.

- 1.1. Wechseln Sie als Benutzer `student` auf `workstation` in das Arbeitsverzeichnis `/home/student/system-network`.

```
[student@workstation ~]$ cd ~/system-network
[student@workstation system-network]$
```

- 1.2. Verifizieren Sie, dass `servera.lab.example.com` Teil der Hostgruppe `webservers` ist. Dieser Server verfügt über eine Ersatz-Netzwerkschnittstelle.

```
[student@workstation system-network]$ cat inventory
[webservers]
servera.lab.example.com
```

- 2. Verifizieren Sie mit dem Befehl `ansible-galaxy`, dass die Systemrollen jetzt verfügbar sind. Wenn keine Rollen verfügbar sind, müssen Sie das Paket `rhel-system-roles` installieren.

```
[student@workstation system-network]$ ansible-galaxy list
# /usr/share/ansible/roles
- linux-system-roles.kdump, (unknown version)
- linux-system-roles.network, (unknown version)
- linux-system-roles.postfix, (unknown version)
- linux-system-roles.selinux, (unknown version)
- linux-system-roles.timesync, (unknown version)
- rhel-system-roles.kdump, (unknown version)
```

```
- rhel-system-roles.network, (unknown version)
- rhel-system-roles.postfix, (unknown version)
- rhel-system-roles.selinux, (unknown version)
- rhel-system-roles.timesync, (unknown version)
# /etc/ansible/roles
[WARNING]: - the configured path /home/student/.ansible/roles does not exist.
```

- 3. Erstellen Sie ein Playbook, das die Rolle „linux-system-roles.network“ zum Konfigurieren der Ersatz-Netzwerkschnittstelle `eth1` mit der IP-Adresse `172.25.250.30` auf `servera.lab.example.com` verwendet.

- 3.1. Erstellen Sie das Playbook `playbook.yml` mit einem Play, dessen Ziel die Hostgruppe `webservers` ist. Nehmen Sie die Rolle `rhel-system-roles.network` in den Abschnitt `roles` des Plays auf.

```
---
- name: NIC Configuration
hosts: webservers

roles:
- rhel-system-roles.network
```

- 3.2. Sehen Sie sich den Abschnitt *Role Variables* für die Rolle `rhel-system-roles.network` in der Datei `README.md` an. Bestimmen Sie die Rollenvariablen zum Konfigurieren der Netzwerkschnittstelle `eth1` mit der IP-Adresse `172.25.250.30`.

```
[student@workstation system-network]$ cat \
> /usr/share/doc/rhel-system-roles/network/README.md
...output omitted...
Setting the IP configuration:
...output omitted...
```

- 3.3. Erstellen Sie das Unterverzeichnis `group_vars/webservers`.

```
[student@workstation system-network]$ mkdir -pv group_vars/webservers
mkdir: created directory 'group_vars'
mkdir: created directory 'group_vars/webservers'
```

- 3.4. Erstellen Sie eine neue Datei `network.yml`, um Rollenvariablen zu definieren. Da diese Variablenwerte für die Hosts der Hostgruppe `webservers` gelten, müssen Sie diese Datei im Verzeichnis `group_vars/webservers` erstellen. Fügen Sie Variablendefinitionen hinzu, um die Konfiguration der Netzwerkschnittstelle `eth1` zu unterstützen. Die Datei enthält jetzt Folgendes:

```
---
network_connections:
- name: eth1
  type: ethernet
  ip:
    address:
      - 172.25.250.30/24
```

- 3.5. Führen Sie das Playbook aus, um die sekundäre Netzwerkschnittstelle auf **servera** zu konfigurieren.

```
[student@workstation system-network]$ ansible-playbook playbook.yml

PLAY [NIC Configuration] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [rhel-system-roles.network : Check which services are running] ****
ok: [servera.lab.example.com]

TASK [rhel-system-roles.network : Check which packages are installed] ****
ok: [servera.lab.example.com]

TASK [rhel-system-roles.network : Print network provider] ****
ok: [servera.lab.example.com] => {
    "msg": "Using network provider: nm"
}

TASK [rhel-system-roles.network : Install packages] ****
skipping: [servera.lab.example.com]

TASK [rhel-system-roles.network : Enable network service] ****
ok: [servera.lab.example.com]

TASK [rhel-system-roles.network : Configure networking connection profiles] ****
...output omitted...

changed: [servera.lab.example.com]

TASK [rhel-system-roles.network : Re-test connectivity] ****
ok: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com      : ok=7      changed=1      unreachable=0      failed=0
skipped=1      rescued=0      ignored=0
```

- 4. Verwenden Sie das Ansible-Modul **setup** in einem Ansible-Ad-hoc-Befehl, um zu verifizieren, dass die Konfiguration der Netzwerkschnittstelle **eth1** auf **servera** korrekt ist.

- 4.1. Verwenden Sie das Ansible-Modul **setup**, um alle Ansible-Fakten für **servera** aufzulisten. Filtern Sie die Ergebnisse für die Netzwerkschnittstelle **eth1** mit der Option **-a 'filter=filter\_string'**. Verifizieren Sie, dass die Netzwerkschnittstelle **eth1** die IP-Adresse **172.25.250.30** verwendet. Die Konfiguration der IP-Adresse kann bis zu einer Minute dauern.

```
[student@workstation system-network]$ ansible webservers -m setup \
> -a 'filter=ansible_eth1'
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_eth1": {
            ...output omitted...
```

```
"ipv4": {  
    "address": "172.25.250.30",  
    "broadcast": "172.25.250.255",  
    "netmask": "255.255.255.0",  
    "network": "172.25.250.0"  
},  
...output omitted...
```

## Beenden

Führen Sie auf `workstation` das Skript `lab system-network finish` aus, um die in dieser Übung erstellten Ressourcen zu bereinigen.

```
[student@workstation ~]$ lab system-network finish
```

Hiermit ist die angeleitete Übung beendet.

## ► Praktische Übung

# Automatisieren von Linux-Administrationsaufgaben

### Leistungscheckliste

In diesem Lab konfigurieren und führen Sie administrative Aufgaben auf verwalteten Hosts mit einem Playbook aus.

### Ergebnisse

Sie sollten in der Lage sein, Playbooks zum Konfigurieren eines Software-Repositorys, von Benutzern und Gruppen, logischen Volumes, cron-Jobs und zusätzlichen Netzwerkschnittstellen auf einem verwalteten Host zu erstellen.

### Bevor Sie Beginnen

Führen Sie auf `workstation` das Start-Skript für die praktische Übung aus, um zu prüfen, ob die Umgebung für die praktische Übung bereit ist. Das Skript erstellt das Arbeitsverzeichnis `system-review` und füllt es mit einer Ansible-Konfigurationsdatei, dem Hostinventar und Dateien für die praktische Übung.

```
[student@workstation ~]$ lab system-review start
```

### Anweisungen

1. Erstellen Sie auf der Hostgruppe `webservers` ein Playbook, das das interne Yum-Repository unter `http://materials.example.com/yum/repository` konfiguriert und das Paket `example-motd` in diesem Repository installiert, und führen Sie es aus. Alle RPM-Pakete werden mit einem GPG-Schlüsselpaar der Organisation signiert. Der öffentliche GPG-Schlüssel befindet sich unter `http://materials.example.com/yum/repository/RPM-GPG-KEY-example`.
2. Erstellen Sie auf der Hostgruppe `webservers` ein Playbook, das die Benutzergruppe `webadmin` erstellt und zwei Benutzer zu dieser Gruppe, `ops1` und `ops2`, hinzufügt, und führen Sie es aus.
3. Erstellen Sie auf der Hostgruppe `webservers` ein Playbook, das das Gerät `/dev/vdb` zum Erstellen einer Volumegruppe namens `apache-vg` verwendet. Dieses Playbook erstellt auch zwei logische Volumes namens `content-lv` und `logs-lv`, die beide von der Volumegruppe `apache-vg` unterstützt werden. Schließlich erstellt es ein XFS-Dateisystem auf jedem logischen Volume und stellt das logische Volume `content-lv` unter `/var/www` und das logische Volume `logs-lv` unter `/var/log/httpd` bereit. Das Lab-Skript füllt zwei Dateien in `~/system-review/storage.yml`, die ein erstes Gerüst für das Playbook bereitstellt, und `storage_vars.yml`, die Werte für alle Variablen liefert, die von den verschiedenen Modulen benötigt werden.
4. Erstellen Sie auf der Hostgruppe `webservers` ein Playbook, das das Modul `cron` verwendet, um die crontab-Datei `/etc/cron.d/disk_usage` zu erstellen, die einen wiederkehrenden cron-Job plant, und führen Sie es aus. Der Job sollte alle zwei Minuten als Benutzer `devops` zwischen 09:00 und 16:59 Uhr von Montag bis Freitag ausgeführt

werden. Der Job sollte die aktuelle Laufwerkverwendung an die Datei /home/devops/disk\_usage anhängen.

5. Erstellen Sie auf der Hostgruppe webserver ein Playbook, das die Rolle `linux-system-roles.network` zum Konfigurieren der Ersatz-Netzwerkschnittstelle `eth1` mit der IP-Adresse `172.25.250.40/24` verwendet, und führen Sie es aus.

## Bewertung

Führen Sie auf `workstation` den Befehl `lab system-review grade` aus, um Ihre Arbeit zu bewerten.

```
[student@workstation ~]$ lab system-review grade
```

## Beenden

Führen Sie auf `workstation` das Skript `lab system-review finish` aus, um die in dieser praktischen Übung erstellten Ressourcen zu bereinigen.

```
[student@workstation ~]$ lab system-review finish
```

Hiermit ist die praktische Übung abgeschlossen.

## ► Lösung

# Automatisieren von Linux-Administrationsaufgaben

### Leistungscheckliste

In diesem Lab konfigurieren und führen Sie administrative Aufgaben auf verwalteten Hosts mit einem Playbook aus.

### Ergebnisse

Sie sollten in der Lage sein, Playbooks zum Konfigurieren eines Software-Repositorys, von Benutzern und Gruppen, logischen Volumes, cron-Jobs und zusätzlichen Netzwerkschnittstellen auf einem verwalteten Host zu erstellen.

### Bevor Sie Beginnen

Führen Sie auf `workstation` das Start-Skript für die praktische Übung aus, um zu prüfen, ob die Umgebung für die praktische Übung bereit ist. Das Skript erstellt das Arbeitsverzeichnis `system-review` und füllt es mit einer Ansible-Konfigurationsdatei, dem Hostinventar und Dateien für die praktische Übung.

```
[student@workstation ~]$ lab system-review start
```

### Anweisungen

1. Erstellen Sie auf der Hostgruppe `webservers` ein Playbook, das das interne Yum-Repository unter `http://materials.example.com/yum/repository` konfiguriert und das Paket `example-motd` in diesem Repository installiert, und führen Sie es aus. Alle RPM-Pakete werden mit einem GPG-Schlüsselpaar der Organisation signiert. Der öffentliche GPG-Schlüssel befindet sich unter `http://materials.example.com/yum/repository/RPM-GPG-KEY-example`.
  - 1.1. Wechseln Sie als Benutzer `student` auf `workstation` in das Arbeitsverzeichnis / `home/student/system-review`.

```
[student@workstation ~]$ cd ~/system-review  
[student@workstation system-review]$
```

- 1.2. Erstellen Sie das Playbook `repo_playbook.yml`, das auf den verwalteten Hosts in der Hostgruppe `webservers` ausgeführt wird. Fügen Sie eine Aufgabe hinzu, die das Modul `yum_repository` verwendet, um die Konfiguration des internen Yum-Repositorys auf dem Remote-Host zu gewährleisten. Stellen Sie Folgendes sicher:
  - Die Konfiguration des Repositorys wird in der Datei `/etc/yum.repos.d/example.repo` gespeichert.
  - Die Repository-ID lautet `example-internal`.
  - Die Basis-URL lautet `http://materials.example.com/yum/repository`.

**Kapitel 9 |** Automatisieren von Linux-Administrationsaufgaben

- Das Repository ist für die Überprüfung von RPM-GPG-Signaturen konfiguriert.
- Die Beschreibung des Repositorys lautet `Example Inc. Internal YUM repo`.

Das Playbook weist folgenden Inhalt auf:

```
---
- name: Repository Configuration
  hosts: webservers
  tasks:
    - name: Ensure Example Repo exists
      yum_repository:
        name: example-internal
        description: Example Inc. Internal YUM repo
        file: example
        baseurl: http://materials.example.com/yum/repository/
        gpgcheck: yes
```

- 1.3. Fügen Sie eine zweite Aufgabe zum Play hinzu, die das Modul `rpm_key` verwendet, um sicherzustellen, dass der öffentliche Schlüssel des Repositorys auf dem Remote-Host vorhanden ist. Die URL des öffentlichen Schlüssels für das Repository lautet `http://materials.example.com/yum/repository/RPM-GPG-KEY-example`.

Die zweite Aufgabe sieht wie folgt aus:

```
- name: Ensure Repo RPM Key is Installed
  rpm_key:
    key: http://materials.example.com/yum/repository/RPM-GPG-KEY-example
    state: present
```

- 1.4. Fügen Sie eine dritte Aufgabe hinzu, um das Paket `example-motd` im internen Yum-Repository zu installieren.

Die dritte Aufgabe sieht wie folgt aus:

```
- name: Install Example motd package
  yum:
    name: example-motd
    state: present
```

- 1.5. Führen Sie das folgende Playbook aus:

```
[student@workstation system-review]$ ansible-playbook repo_playbook.yml

PLAY [Repository Configuration] ****
TASK [Gathering Facts] ****
ok: [serverb.lab.example.com]

TASK [Ensure Example Repo exists] ****
changed: [serverb.lab.example.com]

TASK [Ensure Repo RPM Key is Installed] ****
changed: [serverb.lab.example.com]
```

```
TASK [Install Example motd package] ****
changed: [serverb.lab.example.com]

PLAY RECAP ****
serverb.lab.example.com : ok=4    changed=3    unreachable=0    failed=0
```

2. Erstellen Sie auf der Hostgruppe `webservers` ein Playbook, das die Benutzergruppe `webadmin` erstellt und zwei Benutzer zu dieser Gruppe, `ops1` und `ops2`, hinzufügt, und führen Sie es aus.
- 2.1. Erstellen Sie die Variablendatei `vars/users_vars.yml`, die zwei Benutzer, `ops1` und `ops2`, definiert, die zur Benutzergruppe `webadmin` gehören. Sie müssen evtl. das Unterverzeichnis `vars` erstellen.

```
[student@workstation system-review]$ mkdir vars
[student@workstation system-review]$ vi vars/users_vars.yml
---
users:
  - username: ops1
    groups: webadmin
  - username: ops2
    groups: webadmin
```

- 2.2. Erstellen Sie das Playbook `users.yml`. Definieren Sie ein einzelnes Play im Playbook, das auf die Hostgruppe `webservers` ausgerichtet ist. Fügen Sie die Klausel `vars_files` hinzu, die die Position des Dateinamens `vars/users_vars.yml` definiert. Fügen Sie eine Aufgabe hinzu, die das Modul `group` verwendet, um die Benutzergruppe `webadmin` auf dem Remote-Host zu erstellen.

```
---
- name: Create multiple local users
hosts: webservers
vars_files:
  - vars/users_vars.yml
tasks:
  - name: Add webadmin group
    group:
      name: webadmin
      state: present
```

- 2.3. Fügen Sie dem Playbook eine zweite Aufgabe hinzu, die das Modul `user` zum Erstellen der Benutzer verwendet. Fügen Sie die Klausel `loop: "{{ users }}"` zur Aufgabe hinzu, um die Variablenliste für jeden Benutzernamen in der Datei `vars/users_vars.yml` zu durchlaufen. Verwenden Sie als `name:` für die Benutzer `item.username` als Variablennamen. Auf diese Weise enthält die Variablenliste möglicherweise zusätzliche Informationen, die zum Erstellen der Benutzer hilfreich sein können, z. B. die Gruppen, denen die Benutzer angehören sollen. Die zweite Aufgabe enthält Folgendes:

```
- name: Create user accounts
  user:
    name: "{{ item.username }}"
    groups: webadmin
  loop: "{{ users }}"
```

2.4. Führen Sie das folgende Playbook aus:

```
[student@workstation system-review]$ ansible-playbook users.yml

PLAY [Create multiple local users] ****

TASK [Gathering Facts] ****
ok: [serverb.lab.example.com]

TASK [Add webadmin group] ****
changed: [serverb.lab.example.com]

TASK [Create user accounts] ****
changed: [serverb.lab.example.com] => (item={'username': 'ops1', 'groups': 'webadmin'})
changed: [serverb.lab.example.com] => (item={'username': 'ops2', 'groups': 'webadmin'})

PLAY RECAP ****
serverb.lab.example.com      : ok=3      changed=2      unreachable=0      failed=0
```

3. Erstellen Sie auf der Hostgruppe `webservers` ein Playbook, das das Gerät `/dev/vdb` zum Erstellen einer Volumegruppe namens `apache-vg` verwendet. Dieses Playbook erstellt auch zwei logische Volumes namens `content-lv` und `logs-lv`, die beide von der Volumegruppe `apache-vg` unterstützt werden. Schließlich erstellt es ein XFS-Dateisystem auf jedem logischen Volume und stellt das logische Volume `content-lv` unter `/var/www` und das logische Volume `logs-lv` unter `/var/log/httpd` bereit. Das Lab-Skript füllt zwei Dateien in `~/system-review`: `storage.yml`, die ein erstes Gerüst für das Playbook bereitstellt, und `storage_vars.yml`, die Werte für alle Variablen liefert, die von den verschiedenen Modulen benötigt werden.

3.1. Überprüfen Sie die Variablendatei `storage_vars.yml`.

```
[student@workstation system-review]$ cat storage_vars.yml
---

partitions:
  - number: 1
    start: 1MiB
    end: 257MiB

volume_groups:
  - name: apache-vg
    devices: /dev/vdb1

logical_volumes:
  - name: content-lv
```

```
size: 64M
vgroup: apache-vg
mount_path: /var/www

- name: logs-lv
  size: 128M
  vgroup: apache-vg
  mount_path: /var/log/httpd
```

Diese Datei beschreibt die vorgesehene Struktur von Partitionen, Volumegruppen und logischen Volumes auf den einzelnen Webservern. Die erste Partition beginnt bei einem Offset von 1 MiB vom Anfang des Geräts `/dev/vdb` und endet bei einem Offset von 257 MiB bei einer Gesamtgröße von 256 MiB.

Jeder Webserver verfügt über eine Volumegruppe namens `apache-vg`, die die erste Partition des Geräts `/dev/vdb` enthält.

Jeder Webserver verfügt über zwei logische Volumes. Das erste logische Volume erhält die Bezeichnung `content-lv` und eine Größe von 64 MiB und wird an die Volumegruppe `apache-vg` angehängt sowie bei `/var/www` gemountet. Das zweite logische Volume erhält die Bezeichnung `logs-lv` und eine Größe von 128 MiB und wird an die Volumegruppe `apache-vg` angehängt sowie bei `/var/log/httpd` gemountet.



### Anmerkung

Die Volumegruppe `apache-vg` besitzt eine Kapazität von 256 MiB, da sie von der Partition `/dev/vdb1` unterstützt wird. Sie bietet ausreichend Kapazität für beide logischen Volumes.

- 3.2. Ändern Sie die erste Aufgabe im Playbook `storage.yml`, damit sie das Modul `parted` zum Konfigurieren einer Partition für die einzelnen Loopelemente verwendet. Jedes Element beschreibt eine geplante Partition des Geräts `/dev/vdb` auf den einzelnen Webservern:

#### **number**

Die Partitionsnummer. Verwenden Sie dieses Element als Wert des Keywords `number` für das Modul `parted`.

#### **start**

Der Anfang der Partition als Offset vom Anfang des Blockgeräts. Verwenden Sie dieses Element als Wert des Keywords `part_start` für das Modul `parted`.

#### **end**

Das Ende der Partition als Offset vom Anfang des Blockgeräts. Verwenden Sie dieses Element als Wert des Keywords `part_end` für das Modul `parted`.

Die erste Aufgabe sollte folgenden Inhalt aufweisen:

```
- name: Correct partitions exist on /dev/vdb
parted:
  device: /dev/vdb
  state: present
  number: "{{ item.number }}"
  part_start: "{{ item.start }}"
  part_end: "{{ item.end }}"
loop: "{{ partitions }}"
```

- 3.3. Ändern Sie die zweite Aufgabe des Plays, um das Modul `lvg` zum Konfigurieren einer Volumegruppe für die einzelnen Loopelemente zu verwenden. Jedes Element der Variablen `volume_groups` beschreibt eine Volumegruppe, die auf jedem Webserver vorhanden sein sollte:

**name**

Der Name der Volumegruppe. Verwenden Sie dieses Element als Wert des Keywords `vg` für das Modul `lvg`.

**devices**

Eine durch Komma getrennte Liste von Geräten oder Partitionen, die die Volumegruppe bilden. Verwenden Sie dieses Element als Wert des Keywords `pvs` für das Modul `lvg`.

Die zweite Aufgabe sollte folgenden Inhalt aufweisen:

```
- name: Ensure Volume Groups Exist
lvg:
  vg: "{{ item.name }}"
  pvs: "{{ item.devices }}"
loop: "{{ volume_groups }}"
```

- 3.4. Ändern Sie die dritte Aufgabe, damit diese das Modul `lvol` verwendet. Legen Sie den Namen der Volumegruppe sowie den Namen und die Größe des logischen Volumes mithilfe der Keywords der einzelnen Elemente fest. Die dritte Aufgabe umfasst jetzt den folgenden Inhalt:

```
- name: Create each Logical Volume (LV) if needed
lvol:
  vg: "{{ item.vgroup }}"
  lv: "{{ item.name }}"
  size: "{{ item.size }}"
loop: "{{ logical_volumes }}"
```

- 3.5. Ändern Sie die vierte Aufgabe, um das Modul `filesystem` zu verwenden. Konfigurieren Sie die Aufgabe, um sicherzustellen, dass jedes logische Volume als XFS-Dateisystem formatiert ist. Denken Sie daran, dass ein logisches Volume dem logischen Gerät `/dev/<Name der Volume-Gruppe>/<Name des logischen Volumes>` zugeordnet ist.

Die vierte Aufgabe sollte folgenden Inhalt aufweisen:

```
- name: Ensure XFS Filesystem exists on each LV
  filesystem:
    dev: "/dev/{{ item.vgroup }}/{{ item.name }}"
    fstype: xfs
  loop: "{{ logical_volumes }}"
```

- 3.6. Konfigurieren Sie die fünfte Aufgabe, um sicherzustellen, dass jedes logische Volume über die richtige Storage-Kapazität verfügt. Wenn die Kapazität des logischen Volumen zunimmt, muss das Dateisystem des Volumes erweitert werden.



### Warnung

Wenn die Kapazität eines logischen Volumes verringert werden muss, tritt bei dieser Aufgabe ein Fehler auf, da eine abnehmende Kapazität vom XFS-Dateisystem nicht unterstützt wird.

Die fünfte Aufgabe sollte folgenden Inhalt aufweisen:

```
- name: Ensure the correct capacity for each LV
  lvol:
    vg: "{{ item.vgroup }}"
    lv: "{{ item.name }}"
    size: "{{ item.size }}"
    resizefs: yes
    force: yes
  loop: "{{ logical_volumes }}"
```

- 3.7. Stellen Sie mithilfe des Moduls `mount` in der sechsten Aufgabe sicher, dass jedes logische Volume im entsprechenden Mount-Pfad bereitgestellt wird und nach einem Reboot bestehen bleibt.

Die sechste Aufgabe sollte folgenden Inhalt aufweisen:

```
- name: Each Logical Volume is mounted
  mount:
    path: "{{ item.mount_path }}"
    src: "/dev/{{ item.vgroup }}/{{ item.name }}"
    fstype: xfs
    state: mounted
  loop: "{{ logical_volumes }}"
```

- 3.8. Führen Sie das Playbook aus, um die logischen Volumes auf dem Remote-Host zu erstellen.

```
[student@workstation system-review]$ ansible-playbook storage.yml
PLAY [Ensure Apache Storage Configuration] ****
TASK [Gathering Facts] ****
ok: [serverb.lab.example.com]

TASK [Correct partitions exist on /dev/vdb] ****
```

```

changed: [serverb.lab.example.com] => (item={'number': 1, 'start': '1MiB', 'end': '257MiB'})

TASK [Ensure Volume Groups Exist] ****
changed: [serverb.lab.example.com] => (item={'name': 'apache-vg', 'devices': '/dev/vdb1'})
...output omitted...

TASK [Create each Logical Volume (LV) if needed] ****
changed: [serverb.lab.example.com] => (item={'name': 'content-lv', 'size': '64M', 'vgroup': 'apache-vg', 'mount_path': '/var/www'})
changed: [serverb.lab.example.com] => (item={'name': 'logs-lv', 'size': '128M', 'vgroup': 'apache-vg', 'mount_path': '/var/log/httpd'})

TASK [Ensure XFS Filesystem exists on each LV] ****
changed: [serverb.lab.example.com] => (item={'name': 'content-lv', 'size': '64M', 'vgroup': 'apache-vg', 'mount_path': '/var/www'})
changed: [serverb.lab.example.com] => (item={'name': 'logs-lv', 'size': '128M', 'vgroup': 'apache-vg', 'mount_path': '/var/log/httpd'})

TASK [Ensure the correct capacity for each LV] ****
ok: [serverb.lab.example.com] => (item={'name': 'content-lv', 'size': '64M', 'vgroup': 'apache-vg', 'mount_path': '/var/www'})
ok: [serverb.lab.example.com] => (item={'name': 'logs-lv', 'size': '128M', 'vgroup': 'apache-vg', 'mount_path': '/var/log/httpd'})

TASK [Each Logical Volume is mounted] ****
changed: [serverb.lab.example.com] => (item={'name': 'content-lv', 'size': '64M', 'vgroup': 'apache-vg', 'mount_path': '/var/www'})
changed: [serverb.lab.example.com] => (item={'name': 'logs-lv', 'size': '128M', 'vgroup': 'apache-vg', 'mount_path': '/var/log/httpd'})

PLAY RECAP ****
serverb.lab.example.com      : ok=7      changed=5      unreachable=0      failed=0

```

4. Erstellen Sie auf der Hostgruppe `webservers` ein Playbook, das das Modul `cron` verwendet, um die crontab-Datei `/etc/cron.d/disk_usage` zu erstellen, die einen wiederkehrenden cron-Job plant, und führen sie es aus. Der Job sollte alle zwei Minuten als Benutzer `devops` zwischen 09:00 und 16:59 Uhr von Montag bis Freitag ausgeführt werden. Der Job sollte die aktuelle Laufwerkverwendung an die Datei `/home/devops/disk_usage` anhängen.
  - 4.1. Erstellen Sie das neue Playbook `create_crontab_file.yml`, und fügen Sie die Zeilen hinzu, die für den Start des Plays benötigt werden. Es sollte die verwalteten Hosts in der Gruppe `webservers` als Ziel verwenden und die Rechteerweiterung aktivieren.

```

---
- name: Recurring cron job
  hosts: webservers
  become: true

```

- 4.2. Definieren Sie eine Aufgabe, die das Modul `cron` verwendet, um eine wiederkehrende Cron-Aufgabe zu planen.



### Anmerkung

Das Modul cron bietet eine `name`-Option, um den crontab-Dateieintrag eindeutig zu beschreiben und die erwarteten Ergebnisse sicherzustellen. Die Beschreibung wird der crontab-Datei hinzugefügt. Die Option `name` ist z. B. erforderlich, wenn Sie einen crontab-Eintrag mit `state=absent` entfernen. Darüber hinaus verhindert, wenn der Standardzustand `state=present` festgelegt ist, die Option `name`, dass unabhängig von den bereits vorhandenen Einträgen immer ein neuer crontab-Eintrag erstellt wird.

```
tasks:
  - name: Crontab file exists
    cron:
      name: Add date and time to a file
```

- 4.3. Konfigurieren Sie den Job so, dass er alle zwei Minuten zwischen 09:00 und 16:59 Uhr von Montag bis Freitag ausgeführt wird.

```
minute: "*/2"
hour: 9-16
weekday: 1-5
```

- 4.4. Verwenden Sie den Parameter `cron_file` für die Verwendung der crontab-Datei /etc/cron.d/disk\_usage anstelle der crontab-Datei eines einzelnen Benutzers in /var/spool/cron/. Ein relativer Pfad positioniert die Datei im Verzeichnis /etc/cron.d. Wenn der Parameter `cron_file` verwendet wird, müssen Sie auch den Parameter `user` angeben.

```
user: devops
job: df >> /home/devops/disk_usage
cron_file: disk_usage
state: present
```

- 4.5. Wenn Sie fertig sind, sollte das Playbook wie folgt aussehen: Überprüfen Sie das Playbook auf Fehlerfreiheit.

```
---
- name: Recurring cron job
  hosts: webservers
  become: true

  tasks:
    - name: Crontab file exists
      cron:
        name: Add date and time to a file
        minute: "*/2"
        hour: 9-16
        weekday: 1-5
        user: devops
```

```
job: df >> /home/devops/disk_usage
cron_file: disk_usage
state: present
```

- 4.6. Führen Sie das Playbook aus.

```
[student@workstation system-review]$ ansible-playbook create_crontab_file.yml
PLAY [Recurring cron job] ****
TASK [Gathering Facts] ****
ok: [serverb.lab.example.com]

TASK [Crontab file exists] ****
changed: [serverb.lab.example.com]

PLAY RECAP ****
serverb.lab.example.com : ok=2    changed=1    unreachable=0    failed=0
```

5. Erstellen Sie auf der Hostgruppe `webservers` ein Playbook, das die Rolle `linux-system-roles.network` zum Konfigurieren der Ersatz-Netzwerkschnittstelle `eth1` mit der IP-Adresse `172.25.250.40/24` verwendet, und führen Sie es aus.

- 5.1. Verifizieren Sie mit dem Befehl `ansible-galaxy`, dass die Systemrollen verfügbar sind. Wenn nicht, müssen Sie das Paket `rhel-system-roles` installieren.

```
[student@workstation system-review]$ ansible-galaxy list
# /usr/share/ansible/roles
- linux-system-roles.kdump, (unknown version)
- linux-system-roles.network, (unknown version)
- linux-system-roles.postfix, (unknown version)
- linux-system-roles.selinux, (unknown version)
- linux-system-roles.timesync, (unknown version)
- rhel-system-roles.kdump, (unknown version)
- rhel-system-roles.network, (unknown version)
- rhel-system-roles.postfix, (unknown version)
- rhel-system-roles.selinux, (unknown version)
- rhel-system-roles.timesync, (unknown version)
# /etc/ansible/roles
[WARNING]: - the configured path /home/student/.ansible/roles does not exist.
```

- 5.2. Erstellen Sie das Playbook `network_playbook.yml` mit einem Play, dessen Ziel die Hostgruppe `webservers` ist. Nehmen Sie die Rolle `rhel-system-roles.network` in den Abschnitt `roles` des Plays auf.

```
---
- name: NIC Configuration
  hosts: webservers

  roles:
    - rhel-system-roles.network
```

- 5.3. Erstellen Sie das Unterverzeichnis `group_vars/webservers`.

```
[student@workstation system-review]$ mkdir -pv group_vars/webservers
mkdir: created directory 'group_vars'
mkdir: created directory 'group_vars/webservers'
```

- 5.4. Erstellen Sie eine neue Datei `network.yml`, um Rollenvariablen zu definieren. Da diese Variablenwerte für die Hosts der Hostgruppe `webservers` gelten, müssen Sie diese Datei im Verzeichnis `group_vars/webservers` erstellen. Fügen Sie Variablendefinitionen hinzu, um die Konfiguration der Netzwerkschnittstelle `eth1` zu unterstützen. Die Datei enthält jetzt Folgendes:

```
[student@workstation system-review]$ vi group_vars/webservers/network.yml
---
network_connections:
  - name: eth1
    type: ethernet
    ip:
      address:
        - 172.25.250.40/24
```

- 5.5. Führen Sie das Playbook aus, um die sekundäre Netzwerkschnittstelle zu konfigurieren.

```
[student@workstation system-review]$ ansible-playbook network_playbook.yml

PLAY [NIC Configuration] ****
TASK [Gathering Facts] ****
ok: [serverb.lab.example.com]

TASK [rhel-system-roles.network : Check which services are running] ****
ok: [serverb.lab.example.com]

TASK [rhel-system-roles.network : Check which packages are installed] ****
ok: [serverb.lab.example.com]

TASK [rhel-system-roles.network : Print network provider] ****
ok: [serverb.lab.example.com] => {
    "msg": "Using network provider: nm"
}

TASK [rhel-system-roles.network : Install packages] ****
skipping: [serverb.lab.example.com]

TASK [rhel-system-roles.network : Enable network service] ****
ok: [serverb.lab.example.com]

TASK [rhel-system-roles.network : Configure networking connection profiles] ***
[WARNING]: [002] <info> #0, state:None persistent_state:present, 'eth1': add
connection
eth1, 38d63afed-e610-4929-ba1b-1d38413219fb

changed: [serverb.lab.example.com]
```

## Kapitel 9 | Automatisieren von Linux-Administrationsaufgaben

```
TASK [rhel-system-roles.network : Re-test connectivity] ****
ok: [serverb.lab.example.com]

PLAY RECAP ****
serverb.lab.example.com      : ok=7      changed=1      unreachable=0      failed=0
```

- 5.6. Verifizieren Sie, dass die Netzwerkschnittstelle `eth1` die IP-Adresse `172.25.250.40` verwendet. Die Konfiguration der IP-Adresse kann bis zu einer Minute dauern.

```
[student@workstation system-review]$ ansible webservers -m setup \
> -a 'filter=ansible_eth1'
serverb.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_eth1": {
...output omitted...
            "ipv4": {
                "address": "172.25.250.40",
                "broadcast": "172.25.250.255",
                "netmask": "255.255.255.0",
                "network": "172.25.250.0"
            },
...output omitted...
```

## Bewertung

Führen Sie auf `workstation` den Befehl `lab system-review grade` aus, um Ihre Arbeit zu bewerten.

```
[student@workstation ~]$ lab system-review grade
```

## Beenden

Führen Sie auf `workstation` das Skript `lab system-review finish` aus, um die in dieser praktischen Übung erstellten Ressourcen zu bereinigen.

```
[student@workstation ~]$ lab system-review finish
```

Hiermit ist die praktische Übung abgeschlossen.

# Zusammenfassung

---

In diesem Kapitel erhielten Sie folgende Informationen:

- Das Modul `yum_repository` konfiguriert ein Yum-Repository auf einem verwalteten Host. Bei Repositorys, die öffentliche Schlüssel verwenden, können Sie verifizieren, dass der Schlüssel mit dem Modul `rpm_key` verfügbar ist.
- Die Module `user` und `group` erstellen Benutzer und Gruppen auf einem verwalteten Host. Sie können autorisierte Schlüssel für einen Benutzer mit dem Modul `authorized_key` konfigurieren.
- Cron-Jobs können auf verwalteten Hosts mit dem Modul `cron` konfiguriert werden.
- Ansible unterstützt die Konfiguration von logischen Volumes mit den Modulen `lvg` und `lvol`. Die Module `parted` und `filesystem` unterstützen jeweils die Partitionierung von Geräten und die Erstellung von Dateisystemen.
- Red Hat Enterprise Linux 8 enthält die Netzwerksystemrolle, die die Konfiguration von Netzwerkschnittstellen auf verwalteten Hosts unterstützt.



## Kapitel 10

# Ausführliche Wiederholung: Linux Automation with Ansible

### Ziel

Demonstrieren der in diesem Kurs angeeigneten Fähigkeiten durch Installieren, Optimieren und Konfigurieren von Ansible für die Verwaltung verwalteter Hosts

### Abschnitte

- Ausführliche Wiederholung

### Praxisteil

- Praktische Übung: Bereitstellen von Ansible
- Praktische Übung: Erstellen von Playbooks
- Praktische Übung: Erstellen von Rollen

# Ausführliche Wiederholung

---

## Ziele

Nach Abschluss dieses Abschnitts sollten Sie in der Lage sein, die in *Red Hat Enterprise Linux Automation with Ansible* erworbenen Kenntnisse zu demonstrieren.

## Wiederholung von Red Hat System Administration III: Linux Automation with Ansible

Bevor Sie mit der ausführlichen Wiederholung für diesen Kurs beginnen, sollten Sie mit den in den jeweiligen Kapiteln behandelten Themen vertraut sein.

Für zusätzliche Übungen stehen Ihnen auch die vorherigen Kapitel dieses Lehrbuchs zur Verfügung.

### Kapitel 1, *Einführung in Ansible*

Beschreiben der grundlegenden Konzepte und der Verwendung von Ansible und Installieren von Ansible über Red Hat Ansible Automation Platform.

- Motivation für die Automatisierung von Linux-Administrationsaufgaben mit Ansible, grundlegende Ansible-Konzepte und grundlegende Architektur von Ansible beschreiben.
- Installieren von Ansible auf einem Kontrollknoten und Beschreiben der Unterschiede zwischen Community Ansible und Red Hat Ansible Automation Platform.

### Kapitel 2, *Implementieren eines Ansible-Playbooks*

Erstellen eines Inventars verwalteter Hosts, Schreiben eines einfachen Ansible-Playbooks und Ausführen des Playbooks, um Aufgaben auf diesen Hosts zu automatisieren.

- Beschreiben der Konzepte von Ansible-Inventaren und Verwalten einer statischen Inventar-Datei.
- Beschreiben, wo Ansible-Konfigurationsdateien gespeichert sind, wie Ansible diese auswählt, und diese bearbeiten, um Änderungen an Standardeinstellungen vorzunehmen.
- Eine einzelne Ansible-Automatisierungsaufgabe mit einem Ad-hoc-Befehl ausführen und einige Anwendungsfälle für Ad-hoc-Befehle erläutern.
- Schreiben eines einfachen Ansible-Playbooks und Ausführen dieses Playbooks mit dem Befehl `ansible-playbook`.
- Schreiben eines Playbooks, das mehrere Plays und die Rechteerweiterung auf Play-Basis verwendet, und effektives Verwenden von `ansible-doc`, um zu lernen, wie Sie neue Module verwenden, um Aufgaben für ein Play zu implementieren.

### Kapitel 3, *Verwalten von Variablen und Fakten*

Playbooks erstellen, die Variablen verwenden, um die Verwaltung des Playbooks und der Fakten zu vereinfachen, um Informationen zu den verwalteten Hosts zu referenzieren.

- Erstellen und Verweisen auf Variablen, die bestimmte Hosts oder Hostgruppen, das Play oder die globale Umgebung betreffen, und Beschreibung der Priorisierung von Variablen.
- Vertrauliche Variablen mit Ansible Vault verschlüsseln und Playbooks ausführen, die auf Vault-kodierte Dateien verweisen.
- Daten zu verwalteten Hosts mithilfe von Ansible-Fakten referenzieren und benutzerdefinierte Fakten zu verwalteten Hosts konfigurieren.

## **Kapitel 4, Implementieren der Aufgabensteuerung**

Aufgabensteuerung, Handler und Aufgabenfehler in Ansible-Playbooks verwalten.

- Verwenden von Loops zum Schreiben effizienter Aufgaben und Verwenden von Bedingungen, um zu steuern, wann Aufgaben ausgeführt werden sollen.
- Implementieren von Aufgaben, die nur ausgeführt wird, wenn eine andere Aufgabe den verwalteten Host ändert.
- Festlegen, was passiert, wenn eine Aufgabe fehlschlägt und unter welchen Bedingungen eine Aufgabe fehlschlägt.

## **Kapitel 5, Bereitstellen von Dateien auf verwalteten Hosts**

Bereitstellen, Verwalten und Anpassen von Dateien auf Hosts, die von Ansible verwaltet werden.

- Dateien auf verwalteten Hosts erstellen, installieren, bearbeiten und entfernen und Berechtigungen, Besitz, SELinux-Kontext und andere Merkmale dieser Dateien verwalten.
- Dateien auf verwalteten Hosts bereitstellen, die mithilfe von Jinja2-Vorlagen angepasst werden.

## **Kapitel 6, Verwalten komplexer Plays und Playbooks**

Schreiben von Playbooks für größere, komplexere Plays und Playbooks.

- Ausgefeilte Host-Pattern erstellen, um Hosts für einen Play- oder Ad-hoc-Befehl effizient auszuwählen.
- Verwalten großer Playbooks durch Importieren oder Einbinden anderer Playbooks oder Aufgaben aus externen Dateien, entweder ohne Vorbedingungen oder auf der Grundlage eines bedingten Tests.

## **Kapitel 7, Vereinfachen von Playbooks mit Rollen**

Ansible-Rollen verwenden, um Playbooks schneller zu entwickeln und Ansible-Codes wiederzuverwenden.

- Beschreiben, was eine Rolle ist, wie sie aufgebaut ist und wie sie in einem Playbook verwendet werden kann.
- Schreiben von Playbooks, die RedHat Enterprise Linux System Roles zum Ausführen von Standardvorgängen nutzen.
- Erstellen einer Rolle im Projektverzeichnis eines Playbooks und diese als Teil eines Plays im Playbook ausführen.
- Auswählen und Abrufen von Rollen aus der Ansible-Galaxy oder anderen Quellen, z. B. einem Git-Repository, und Verwenden in Ihren Playbooks.

## Kapitel 10 | Ausführliche Wiederholung: Linux Automation with Ansible

- Abrufen einer Reihe von verwandten Rollen, Zusatzmodulen und anderen Inhalten aus Content-Sammlungen und Verwenden in einem Playbook.

## Kapitel 8, Fehlerbehebung in Ansible

Behben von Fehlern in Playbooks und auf verwalteten Hosts.

- Beheben von allgemeinen Probleme mit einem neuen Playbook.
- Beheben von Fehlern auf verwalteten Hosts beim Ausführen eines Playbooks.

## Kapitel 9, Automatisieren von Linux-Administrationsaufgaben

Allgemeine Linux-Systemadministrationsaufgaben mit Ansible automatisieren

- Systeme abonnieren, Softwarekanäle und Repositorys konfigurieren und RPM-Pakete auf verwalteten Hosts verwalten.
- Linux-Benutzer und -Gruppen verwalten, SSH konfigurieren und die Sudo-Konfiguration auf verwalteten Hosts ändern.
- Start eines Services verwalten, Prozesse mit at, cron und systemd planen, einen Reboot durchführen und das Standardstartziel auf verwalteten Hosts steuern.
- Speichergeräte partitionieren, LVM konfigurieren, Partitionen oder logische Volumes formatieren, Dateisysteme mounten und Swap-Dateien oder -Bereiche hinzufügen.
- Netzwerkeinstellungen und Namensauflösung für verwaltete Hosts konfigurieren und netzwerkbezogene Ansible-Fakten sammeln.

## ► Praktische Übung

# Bereitstellen von Ansible

In dieser Wiederholung installieren Sie Ansible auf `workstation`, verwenden Ansible als Kontrollknoten und konfigurieren es zum Herstellen einer Verbindung mit den verwalteten Hosts `servera` und `serverb`. Ad-hoc-Befehle zur Durchführung von Aktionen auf verwalteten Hosts verwenden

### Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Ansible installieren
- Ad-hoc-Befehle zur Durchführung von Aktionen auf verwalteten Hosts verwenden

### Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab review-deploy start` aus. Dieses Skript stellt sicher, dass die verwalteten Hosts `servera` und `serverb` im Netzwerk erreichbar sind. Das Skript erstellt das Unterverzeichnis `review-deploy` für die Übung im Benutzerverzeichnis des Teilnehmers.

```
[student@workstation ~]$ lab review-deploy start
```

### Anweisungen

1. Installieren Sie Ansible auf `workstation`, damit der Rechner als Kontrollknoten verwendet werden kann.
2. Erstellen Sie auf dem Kontrollknoten die Inventardatei `/home/student/review-deploy/inventory`, die eine Gruppe mit dem Namen `dev` enthält. Diese Gruppe sollte aus den verwalteten Hosts `servera.lab.example.com` und `serverb.lab.example.com` bestehen.
3. Erstellen Sie die Ansible-Konfigurationsdatei `/home/student/review-deploy/ansible.cfg`. Die Konfigurationsdatei sollte auf die Inventardatei `/home/student/review-deploy/inventory` verweisen.
4. Führen Sie mittels Rechteerweiterung einen Ad-hoc-Befehl aus, um die Inhalte der Datei `/etc/motd` auf `servera` und `serverb` so zu ändern, dass sie die Zeichenfolge `Managed by Ansible\n` enthält. Verwenden Sie `devops` als Remote-Benutzer.
5. Führen Sie einen Ad-hoc-Befehl aus, um sicherzustellen, dass die Inhalte der Datei `/etc/motd` auf `servera` und `serverb` identisch sind.

### Bewertung

Führen Sie auf `workstation` den Befehl `lab review-deploy grade` aus, um den Erfolg dieser Übung zu bestätigen. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie den Befehl so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab review-deploy grade
```

## Beenden

Führen Sie auf workstation den Befehl `lab review-deploy finish` aus, um diese Übung zu bereinigen.

```
[student@workstation ~]$ lab review-deploy finish
```

Hiermit ist die praktische Übung abgeschlossen.

## ► Lösung

# Bereitstellen von Ansible

In dieser Wiederholung installieren Sie Ansible auf `workstation`, verwenden Ansible als Kontrollknoten und konfigurieren es zum Herstellen einer Verbindung mit den verwalteten Hosts `servera` und `serverb`. Ad-hoc-Befehle zur Durchführung von Aktionen auf verwalteten Hosts verwenden

## Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Ansible installieren
- Ad-hoc-Befehle zur Durchführung von Aktionen auf verwalteten Hosts verwenden

## Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab review-deploy start` aus. Dieses Skript stellt sicher, dass die verwalteten Hosts `servera` und `serverb` im Netzwerk erreichbar sind. Das Skript erstellt das Unterverzeichnis `review-deploy` für die Übung im Benutzerverzeichnis des Teilnehmers.

```
[student@workstation ~]$ lab review-deploy start
```

## Anweisungen

1. Installieren Sie Ansible auf `workstation`, damit der Rechner als Kontrollknoten verwendet werden kann.

```
[student@workstation ~]$ sudo yum install ansible
[sudo] password for student:
Loaded plugins: langpacks, search-disabled-repos
Resolving Dependencies
--> Running transaction check
...output omitted...
Is this ok [y/d/N]: y
...output omitted...
```

2. Erstellen Sie auf dem Kontrollknoten die Inventardatei `/home/student/review-deploy/inventory`, die eine Gruppe mit dem Namen `dev` enthält. Diese Gruppe sollte aus den verwalteten Hosts `servera.lab.example.com` und `serverb.lab.example.com` bestehen.
  - 2.1. Wechseln Sie in das Ansible-Projektverzeichnis `/home/student/review-deploy`, das vom Setup-Skript erstellt wurde.

```
[student@workstation ~]$ cd ~/review-deploy
```

2.2. Erstellen Sie die Datei `inventory` mit dem folgenden Inhalt.

```
[dev]
servera.lab.example.com
serverb.lab.example.com
```

3. Erstellen Sie die Ansible-Konfigurationsdatei `/home/student/review-deploy/ansible.cfg`. Die Konfigurationsdatei sollte auf die Inventardatei `/home/student/review-deploy/inventory` verweisen.

Fügen Sie die folgenden Einträge hinzu, um die Inventardatei `./inventory` als Inventarquelle zu konfigurieren. Speichern Sie Ihre Änderungen und beenden Sie den Texteditor.

```
[defaults]
inventory=../inventory
```

4. Führen Sie mittels Rechteerweiterung einen Ad-hoc-Befehl aus, um die Inhalte der Datei `/etc/motd` auf `servera` und `serverb` so zu ändern, dass sie die Zeichenfolge `Managed by Ansible\n` enthält. Verwenden Sie `devops` als Remote-Benutzer.

```
[student@workstation review-deploy]$ ansible dev -m copy \
> -a 'content="Managed by Ansible\n" dest=/etc/motd' -b -u devops
servera.lab.example.com | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": true,
    "checksum": "4458b979ede3c332f8f2128385df4ba305e58c27",
    "dest": "/etc/motd",
    "gid": 0,
    "group": "root",
    "md5sum": "65a4290ee5559756ad04e558b0e0c4e3",
    "mode": "0644",
    "owner": "root",
    "secontext": "unconfined_u:object_r:etc_t:s0",
    "size": 19,
    "src": "/home/devops/.ansible/tmp/...output omitted...",
    "state": "file",
    "uid": 0
}
serverb.lab.example.com | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": true,
    "checksum": "4458b979ede3c332f8f2128385df4ba305e58c27",
    "dest": "/etc/motd",
    "gid": 0,
    "group": "root",
    "md5sum": "65a4290ee5559756ad04e558b0e0c4e3",
    "mode": "0644",
    "owner": "root",
    "secontext": "system_u:object_r:etc_t:s0",
```

```
"size": 19,  
"src": "/home/devops/.ansible/tmp/...output omitted...",  
"state": "file",  
"uid": 0  
}
```

- Führen Sie einen Ad-hoc-Befehl aus, um sicherzustellen, dass die Inhalte der Datei /etc/motd auf servera und serverb identisch sind.

```
[student@workstation review-deploy]$ ansible dev -m command -a "cat /etc/motd"  
servera.lab.example.com | CHANGED | rc=0 >>  
Managed by Ansible  
  
serverb.lab.example.com | CHANGED | rc=0 >>  
Managed by Ansible
```

## Bewertung

Führen Sie auf workstation den Befehl lab review-deploy grade aus, um den Erfolg dieser Übung zu bestätigen. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie den Befehl so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab review-deploy grade
```

## Beenden

Führen Sie auf workstation den Befehl lab review-deploy finish aus, um diese Übung zu bereinigen.

```
[student@workstation ~]$ lab review-deploy finish
```

Hiermit ist die praktische Übung abgeschlossen.

## ► Praktische Übung

# Erstellen von Playbooks

In dieser Wiederholung erstellen Sie im Ansible-Projektverzeichnis `/home/student/review-playbooks` drei Playbooks. Mit einem Playbook wird sichergestellt, dass `lftp` auf Systemen installiert wird, die FTP-Clients sein sollen. Ein weiteres Playbook gewährleistet, dass `vsftpd` auf Systemen installiert und konfiguriert wird, die als FTP-Server fungieren sollen, und mit dem dritten Playbook (`site.yml`) werden die beiden anderen Playbooks ausgeführt.

### Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Playbooks zur Durchführung von Aufgaben auf verwalteten Hosts erstellen und ausführen
- Jinja2-Vorlagen, -Variablen und -Handler in Playbooks verwenden



#### Wichtig

Wenn Sie Schwierigkeiten mit dem Playbook `site.yml` haben, vergewissern Sie sich, dass `ansible-vsftpd.yml` und `ftpclients.yml` eine übereinstimmende Einrückung aufweisen.

### Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab review-playbooks start` aus.

```
[student@workstation ~]$ lab review-playbooks start
```

### Anweisungen

1. Erstellen Sie als Benutzer `student` auf `workstation` die Inventardatei `/home/student/review-playbooks/inventory`, die `serverc.lab.example.com` in der Gruppe `ftpclients` sowie `serverb.lab.example.com` und `serverd.lab.example.com` in der Gruppe `ftpservers` enthält.
2. Erstellen Sie die Ansible-Konfigurationsdatei `/home/student/review-playbooks/ansible.cfg`, und füllen Sie sie mit den erforderlichen Einträgen, um diese Anforderungen zu erfüllen:
  - Konfigurieren des Ansible-Projekts für die Verwendung des neu erstellten Inventars
  - Herstellen einer Verbindung zu verwalteten Hosts als `devops`-Benutzer
  - Verwenden der Rechteerweiterungen mit `sudo` als `root`-Benutzer
  - Standardmäßiges Erweitern der Rechte für jede Aufgabe

3. Erstellen Sie das Playbook `/home/student/review-playbooks/ftpclients.yml`, das ein Play für die Zielhosts in der Inventargruppe `ftpclients` enthält und sicherstellt, dass das `lftp`-Paket installiert ist.
4. Legen Sie die bereitgestellte vsftpd-Konfigurationsdatei `vsftpd.conf.j2` im Unterverzeichnis `templates` ab.
5. Legen Sie die bereitgestellte Datei `defaults-template.yml` im Unterverzeichnis `vars` ab.
6. Erstellen Sie im Unterverzeichnis `vars` die Variablendefinitionsdatei `vars.yml`, um die folgenden drei Variablen und ihre Werte zu definieren:

Variable	Wert
<code>vsftpd_package</code>	<code>vsftpd</code>
<code>vsftpd_service</code>	<code>vsftpd</code>
<code>vsftpd_config_file</code>	<code>/etc/vsftpd/vsftpd.conf</code>

7. Erstellen Sie mithilfe der zuvor erstellten Jinja2-Vorlage und der Variablendefinitionsdateien ein zweites Playbook mit dem Namen `/home/student/review-playbooks/ansible-vsftpd.yml`, um den vsftpd-Service auf den Hosts in der Inventargruppe `ftpservers` zu konfigurieren.
8. Erstellen Sie ein drittes Playbook mit dem Namen `/home/student/review-playbooks/site.yml`, und schließen Sie die Plays der beiden zuvor erstellten Playbooks `ftpclients.yml` und `ansible-vsftpd.yml` darin ein.
9. Führen Sie das Playbook `/home/student/review-playbooks/site.yml` aus, um zu prüfen, ob es die gewünschten Aufgaben auf den verwalteten Hosts durchführt.

## Bewertung

Führen Sie auf `workstation` als Benutzer `student` den Befehl `lab review-playbooks grade` aus, um den Erfolg dieser Übung zu bestätigen. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie das Skript so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab review-playbooks grade
```

## Beenden

Führen Sie den Befehl `lab review-playbooks finish` aus, um die Übungsaufgaben auf `serverb`, `serverc` und `serverd` zu löschen.

```
[student@workstation ~]$ lab review-playbooks finish
```

Hiermit ist die praktische Übung abgeschlossen.

## ► Lösung

# Erstellen von Playbooks

In dieser Wiederholung erstellen Sie im Ansible-Projektverzeichnis `/home/student/review-playbooks` drei Playbooks. Mit einem Playbook wird sichergestellt, dass `lftp` auf Systemen installiert wird, die FTP-Clients sein sollen. Ein weiteres Playbook gewährleistet, dass `vsftpd` auf Systemen installiert und konfiguriert wird, die als FTP-Server fungieren sollen, und mit dem dritten Playbook (`site.yml`) werden die beiden anderen Playbooks ausgeführt.

### Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Playbooks zur Durchführung von Aufgaben auf verwalteten Hosts erstellen und ausführen
- Jinja2-Vorlagen, -Variablen und -Handler in Playbooks verwenden



#### Wichtig

Wenn Sie Schwierigkeiten mit dem Playbook `site.yml` haben, vergewissern Sie sich, dass `ansible-vsftpd.yml` und `ftpclients.yml` eine übereinstimmende Einrückung aufweisen.

### Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab review-playbooks start` aus.

```
[student@workstation ~]$ lab review-playbooks start
```

### Anweisungen

1. Erstellen Sie als Benutzer `student` auf `workstation` die Inventardatei `/home/student/review-playbooks/inventory`, die `serverc.lab.example.com` in der Gruppe `ftpclients` sowie `serverb.lab.example.com` und `serverd.lab.example.com` in der Gruppe `ftpservers` enthält.
  - 1.1. Wechseln Sie in das Ansible-Projektverzeichnis `/home/student/review-playbooks`, das vom Setup-Skript erstellt wurde.

```
[student@workstation ~]$ cd ~/review-playbooks
```

- 1.2. Füllen Sie die Datei `inventory` mit den folgenden Einträgen, speichern und schließen Sie die Datei.

```
[ftpservers]
serverb.lab.example.com
serverd.lab.example.com

[ftpclients]
serverc.lab.example.com
```

2. Erstellen Sie die Ansible-Konfigurationsdatei /home/student/review-playbooks/ansible.cfg, und füllen Sie sie mit den erforderlichen Einträgen, um diese Anforderungen zu erfüllen:

- Konfigurieren des Ansible-Projekts für die Verwendung des neu erstellten Inventars
- Herstellen einer Verbindung zu verwalteten Hosts als devops-Benutzer
- Verwenden der Rechteerweiterungen mit sudo als root-Benutzer
- Standardmäßiges Erweitern der Rechte für jede Aufgabe

```
[defaults]
remote_user = devops
inventory = ./inventory

[privilegeEscalation]
become_user = root
become_method = sudo
become = true
```

3. Erstellen Sie das Playbook /home/student/review-playbooks/ftpclients.yml, das ein Play für die Zielhosts in der Inventargruppe ftpclients enthält und sicherstellt, dass das lftp-Paket installiert ist.

```
---
- name: Ensure FTP Client Configuration
  hosts: ftpclients

  tasks:
    - name: latest version of lftp is installed
      yum:
        name: lftp
        state: latest
```

4. Legen Sie die bereitgestellte vsftpd-Konfigurationsdatei vsftpd.conf.j2 im Unterverzeichnis templates ab.

- 4.1. Erstellen Sie das Unterverzeichnis templates.

```
[student@workstation review-playbooks]$ mkdir -v templates
mkdir: created directory 'templates'
```

- 4.2. Verschieben Sie die Datei vsftpd.conf.j2 in das neu erstellte Unterverzeichnis templates.

```
[student@workstation review-playbooks]$ mv -v vsftpd.conf.j2 templates/
renamed 'vsftpd.conf.j2' -> 'templates/vsftpd.conf.j2'
```

5. Legen Sie die bereitgestellte Datei `defaults-template.yml` im Unterverzeichnis `vars` ab.

5.1. Erstellen Sie das Unterverzeichnis `vars`.

```
[student@workstation review-playbooks]$ mkdir -v vars
mkdir: created directory 'vars'
```

5.2. Verschieben Sie die Datei `defaults-template.yml` in das neu erstellte Unterverzeichnis `vars`.

```
[student@workstation review-playbooks]$ mv -v defaults-template.yml vars/
renamed 'defaults-template.yml' -> 'vars/defaults-template.yml'
```

6. Erstellen Sie im Unterverzeichnis `vars` die Variablendefinitionsdatei `vars.yml`, um die folgenden drei Variablen und ihre Werte zu definieren:

Variable	Wert
<code>vsftpd_package</code>	<code>vsftpd</code>
<code>vsftpd_service</code>	<code>vsftpd</code>
<code>vsftpd_config_file</code>	<code>/etc/vsftpd/vsftpd.conf</code>

```
vsftpd_package: vsftpd
vsftpd_service: vsftpd
vsftpd_config_file: /etc/vsftpd/vsftpd.conf
```

7. Erstellen Sie mithilfe der zuvor erstellten Jinja2-Vorlage und der Variablendefinitionsdateien ein zweites Playbook mit dem Namen `/home/student/review-playbooks/ansible-vsftpd.yml`, um den vsftpd-Service auf den Hosts in der Inventargruppe `ftpservers` zu konfigurieren.

```
---
- name: FTP server is installed
  hosts:
    - ftptests
  vars_files:
    - vars/defaults-template.yml
    - vars/vars.yml

  tasks:
    - name: Packages are installed
      yum:
        name: "{{ vsftpd_package }}"
        state: present
```

```
- name: Ensure service is started
  service:
    name: "{{ vsftpd_service }}"
    state: started
    enabled: true

- name: Configuration file is installed
  template:
    src: templates/vsftpd.conf.j2
    dest: "{{ vsftpd_config_file }}"
    owner: root
    group: root
    mode: 0600
    setype: etc_t
  notify: restart vsftpd

- name: firewalld is installed
  yum:
    name: firewalld
    state: present

- name: firewalld is started and enabled
  service:
    name: firewalld
    state: started
    enabled: yes

- name: FTP port is open
  firewalld:
    service: ftp
    permanent: true
    state: enabled
    immediate: yes

- name: FTP passive data ports are open
  firewalld:
    port: 21000-21020/tcp
    permanent: yes
    state: enabled
    immediate: yes

handlers:
- name: restart vsftpd
  service:
    name: "{{ vsftpd_service }}"
    state: restarted
```

8. Erstellen Sie ein drittes Playbook mit dem Namen `/home/student/review-playbooks/site.yml`, und schließen Sie die Plays der beiden zuvor erstellten Playbooks `ftpclients.yml` und `ansible-vsftpd.yml` darin ein.

```
---  
# FTP Servers playbook  
- import_playbook: ansible-vsftpd.yml  
  
# FTP Clients playbook  
- import_playbook: ftpclients.yml
```

9. Führen Sie das Playbook `/home/student/review-playbooks/site.yml` aus, um zu prüfen, ob es die gewünschten Aufgaben auf den verwalteten Hosts durchführt.

```
[student@workstation review-playbooks]$ ansible-playbook site.yml
```

## Bewertung

Führen Sie auf `workstation` als Benutzer `student` den Befehl `lab review-playbooks grade` aus, um den Erfolg dieser Übung zu bestätigen. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie das Skript so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab review-playbooks grade
```

## Beenden

Führen Sie den Befehl `lab review-playbooks finish`, um die Übungsaufgaben auf `serverb`, `serverc` und `serverd` zu löschen.

```
[student@workstation ~]$ lab review-playbooks finish
```

Hiermit ist die praktische Übung abgeschlossen.

## ► Praktische Übung

### Erstellen von Rollen

In dieser Wiederholung konvertieren Sie das Playbook `ansible-vsftpd.yml` in eine Rolle und verwenden diese Rolle anschließend in einem neuen Playbook, mit dem auch einige zusätzliche Aufgaben durchgeführt werden.

#### Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Erstellen einer Rolle zum Konfigurieren des Service `vsftpd` mit Aufgaben aus einem vorhandenen Playbook
- Einbeziehen einer Rolle in ein Playbook und Ausführen des Playbooks



#### Wichtig

Es könnte nützlich sein, Ihre Rolle zu debuggen, indem Sie sie in einem Playbook testen, das nicht die oben aufgeführten zusätzlichen Aufgaben oder Playbook-Variablen, sondern nur ein Play für die Hosts in der Gruppe `ftpservers` enthält und die Rolle anwendet.

Sobald Sie bestätigt haben, dass ein vereinfachtes Playbook, in dem lediglich die Rolle verwendet wird, genau wie das ursprüngliche Playbook `ansible-vsftpd.yml` funktioniert, können Sie das vollständige Playbook `vsftpd-configure.yml` erstellen, indem Sie die oben angegebenen zusätzlichen Variablen und Aufgaben hinzufügen.



#### Wichtig

Wenn Sie Schwierigkeiten mit dem Playbook `site.yml` haben, vergewissern Sie sich, dass `vsftpd-configure.yml` und `ftpclients.yml` eine übereinstimmende Einrückung aufweisen.

### Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab review-roles start` aus. Dieses Skript stellt sicher, dass die Remote-Hosts im Netzwerk erreichbar sind. Das Setup-Skript überprüft, ob Ansible auf `workstation` installiert ist, erstellt eine Verzeichnisstruktur für die Übungsumgebung und installiert erforderliche Übungsdateien.

```
[student@workstation ~]$ lab review-roles start
```

## Anweisungen

1. Wechseln Sie in das Arbeitsverzeichnis `review-roles`. Konfigurieren Sie das Ansible-Projekt so, dass die statische Inventardatei `inventory` verwendet wird. Verifizieren Sie die Inventarkonfiguration mit dem Befehl `ansible-inventory`.
2. Konvertieren Sie das Playbook `ansible-vsftpd.yml` in die Rolle `ansible-vsftpd`.
3. Aktualisieren Sie den Inhalt der Datei `roles/ansible-vsftpd/meta/main.yml`.

Variable	Wert
Autor	Red Hat Training
description	Beispielrolle für RH294
Unternehmen	Red Hat
Lizenz	BSD

4. Bearbeiten Sie den Inhalt der Datei `roles/ansible-vsftpd/README.md` für die Rolle, damit die Datei relevante Informationen zur Rolle bereitstellt. Nach der Bearbeitung sollte die Datei Folgendes enthalten:

```
ansible-vsftpd
=====
Example ansible-vsftpd role from Red Hat's "Linux Automation" (RH294)
course.

Role Variables
-----
* defaults/main.yml contains variables used to configure the vsftpd.conf template
* vars/main.yml contains the name of the vsftpd service, the name of the RPM
package, and the location of the service's configuration file

Dependencies
-----
None.

Example Playbook
-----
- hosts: servers
  roles:
    - ansible-vsftpd

License
-----
BSD
```

**Author Information**

Red Hat (training@redhat.com)

5. Entfernen Sie die nicht verwendeten Verzeichnisse aus der neuen Rolle.
6. Erstellen Sie das neue Playbook `vsftpd-configure.yml`. Es sollte Folgendes enthalten.

```
---
- name: Install and configure vsftpd
  hosts: ftpservers
  vars:
    vsftpd_anon_root: /mnt/share/
    vsftpd_local_root: /mnt/share/

  roles:
    - ansible-vsftpd

  tasks:
    - name: /dev/vdb1 is partitioned
      parted:
        device: /dev/vdb
        number: 1
        label: gpt
        part_start: 1MiB
        part_end: 100%
        state: present

    - name: XFS file system exists on /dev/vdb1
      filesystem:
        dev: /dev/vdb1
        fstype: xfs
        force: yes

    - name: anon_root mount point exists
      file:
        path: '{{ vsftpd_anon_root }}'
        state: directory

    - name: /dev/vdb1 is mounted on anon_root
      mount:
        path: '{{ vsftpd_anon_root }}'
        src: /dev/vdb1
        fstype: xfs
        state: mounted
        dump: '1'
        passno: '2'
      notify: restart vsftpd

    - name: Make sure permissions on mounted fs are correct
      file:
        path: '{{ vsftpd_anon_root }}'
        owner: root
        group: root
```

```
mode: '0755'  
setype: "{{ vsftpd_setype }}"  
state: directory  
  
- name: Copy README to the ftp anon_root  
  copy:  
    dest: '{{ vsftpd_anon_root }}/README'  
    content: "Welcome to the FTP server at {{ ansible_fqdn }}\n"  
    setype: '{{ vsftpd_setype }}'
```

7. Ändern Sie das Playbook `site.yml` so, dass statt des Playbooks `ansible-vsftpd.yml` das neu erstellte Playbook `vsftpd-configure.yml` verwendet wird.
8. Verifizieren Sie, dass das Playbook `site.yml` wie gewünscht funktioniert, indem Sie es mit `ansible-playbook` ausführen.

## Bewertung

Führen Sie auf `workstation` den Befehl `lab review-roles grade` aus, um den Erfolg dieser Übung zu bestätigen. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie das Skript so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab review-roles grade
```

## Beenden

Führen Sie den Befehl `lab review-roles finish` aus, um die Übungsaufgaben auf `servera` und `serverb` zu löschen.

```
[student@workstation ~]$ lab review-roles finish
```

Hiermit ist die praktische Übung abgeschlossen.

## ► Lösung

# Erstellen von Rollen

In dieser Wiederholung konvertieren Sie das Playbook `ansible-vsftpd.yml` in eine Rolle und verwenden diese Rolle anschließend in einem neuen Playbook, mit dem auch einige zusätzliche Aufgaben durchgeführt werden.

## Ergebnisse

Es werden folgende Fähigkeiten vermittelt:

- Erstellen einer Rolle zum Konfigurieren des Service `vsftpd` mit Aufgaben aus einem vorhandenen Playbook
- Einbeziehen einer Rolle in ein Playbook und Ausführen des Playbooks



### Wichtig

Es könnte nützlich sein, Ihre Rolle zu debuggen, indem Sie sie in einem Playbook testen, das nicht die oben aufgeführten zusätzlichen Aufgaben oder Playbook-Variablen, sondern nur ein Play für die Hosts in der Gruppe `ftpservers` enthält und die Rolle anwendet.

Sobald Sie bestätigt haben, dass ein vereinfachtes Playbook, in dem lediglich die Rolle verwendet wird, genau wie das ursprüngliche Playbook `ansible-vsftpd.yml` funktioniert, können Sie das vollständige Playbook `vsftpd-configure.yml` erstellen, indem Sie die oben angegebenen zusätzlichen Variablen und Aufgaben hinzufügen.



### Wichtig

Wenn Sie Schwierigkeiten mit dem Playbook `site.yml` haben, vergewissern Sie sich, dass `vsftpd-configure.yml` und `ftpclients.yml` eine übereinstimmende Einrückung aufweisen.

## Bevor Sie Beginnen

Melden Sie sich als `student` mit dem Passwort `student` bei `workstation` an.

Führen Sie auf `workstation` den Befehl `lab review-roles start` aus. Dieses Skript stellt sicher, dass die Remote-Hosts im Netzwerk erreichbar sind. Das Setup-Skript überprüft, ob Ansible auf `workstation` installiert ist, erstellt eine Verzeichnisstruktur für die Übungsumgebung und installiert erforderliche Übungsdateien.

```
[student@workstation ~]$ lab review-roles start
```

## Anweisungen

1. Wechseln Sie in das Arbeitsverzeichnis `review-roles`. Konfigurieren Sie das Ansible-Projekt so, dass die statische Inventardatei `inventory` verwendet wird. Verifizieren Sie die Inventarkonfiguration mit dem Befehl `ansible-inventory`.

- 1.1. Wechseln Sie in das Arbeitsverzeichnis `review-roles`.

```
[student@workstation ~]$ cd ~/review-roles
[student@workstation review-roles]$
```

- 1.2. Bearbeiten Sie die Datei `ansible.cfg`, fügen Sie die Direktive `inventory` zum Abschnitt `[defaults]` hinzu, und legen Sie diese auf `./inventory` fest.

Der Abschnitt `[defaults]` der Datei `ansible.cfg` sieht folgendermaßen aus:

```
[defaults]
remote_user=devops
inventory=./inventory
```

- 1.3. Verifizieren Sie mit dem Befehl `ansible-inventory` die Inventarkonfiguration des Projekts:

```
[student@workstation review-roles]$ ansible-inventory --list all
{
    "_meta": {
        "hostvars": {}
    },
    "all": {
        "children": [
            "ftpclients",
            "ftpservers",
            "ungrouped"
        ]
    },
    "ftpclients": {
        "hosts": [
            "servera.lab.example.com",
            "serverc.lab.example.com"
        ]
    },
    "ftpservers": {
        "hosts": [
            "serverb.lab.example.com",
            "serverd.lab.example.com"
        ]
    }
}
```

2. Konvertieren Sie das Playbook `ansible-vsftpd.yml` in die Rolle `ansible-vsftpd`.

- 2.1. Erstellen Sie das Unterverzeichnis `roles`.

```
[student@workstation review-roles]$ mkdir -v roles  
mkdir: created directory 'roles'
```

- 2.2. Erstellen Sie mit `ansible-galaxy` die Verzeichnisstruktur für die neue Rolle `ansible-vsftpd` im Unterverzeichnis `roles`.

```
[student@workstation review-roles]$ cd roles  
[student@workstation roles]$ ansible-galaxy init ansible-vsftpd  
- Role ansible-vsftpd was created successfully  
[student@workstation roles]$ cd ..  
[student@workstation review-roles]$
```

- 2.3. Verifizieren Sie mithilfe von `tree` die Verzeichnisstruktur, die für die neue Rolle erstellt wurde.

```
[student@workstation review-roles]$ tree roles  
roles  
└── ansible-vsftpd  
    ├── defaults  
    │   └── main.yml  
    ├── files  
    ├── handlers  
    │   └── main.yml  
    ├── meta  
    │   └── main.yml  
    ├── README.md  
    ├── tasks  
    │   └── main.yml  
    ├── templates  
    ├── tests  
    │   ├── inventory  
    │   └── test.yml  
    └── vars  
        └── main.yml  
  
9 directories, 8 files
```

- 2.4. Ersetzen Sie die Datei `roles/ansible-vsftpd/defaults/main.yml` durch die Variablendefinitionen in der Datei `defaults-template.yml`.

```
[student@workstation review-roles]$ mv -v defaults-template.yml \  
> roles/ansible-vsftpd/defaults/main.yml  
renamed 'defaults-template.yml' -> 'roles/ansible-vsftpd/defaults/main.yml'
```

- 2.5. Ersetzen Sie die Datei `roles/ansible-vsftpd/vars/main.yml` durch die Variablendefinitionen in der Datei `vars.yml`.

```
[student@workstation review-roles]$ mv -v vars.yml \  
> roles/ansible-vsftpd/vars/main.yml  
renamed 'vars.yml' -> 'roles/ansible-vsftpd/vars/main.yml'
```

- 2.6. Verwenden Sie die Datei `templates/vsftpd.conf.j2` als Vorlage für die Rolle `ansible-vsftpd`.

```
[student@workstation review-roles]$ mv -v vsftpd.conf.j2 \
> roles/ansible-vsftpd/templates/
renamed 'vsftpd.conf.j2' -> 'roles/ansible-vsftpd/templates/vsftpd.conf.j2'
```

- 2.7. Kopieren Sie Aufgaben aus dem Playbook `ansible-vsftpd.yml` in die Datei `roles/ansible-vsftpd/tasks/main.yml`. Der Wert des Keywords `src` in der Modulaufgabe `template` muss nicht mehr auf das Unterverzeichnis `templates` verweisen. Die Datei `roles/ansible-vsftpd/tasks/main.yml` sollte Folgendes enthalten, wenn Sie fertig sind.

```
---
# tasks file for ansible-vsftpd
- name: Packages are installed
  yum:
    name: '{{ vsftpd_package }}'
    state: present

- name: Ensure service is started
  service:
    name: '{{ vsftpd_service }}'
    state: started
    enabled: true

- name: Configuration file is installed
  template:
    src: vsftpd.conf.j2
    dest: '{{ vsftpd_config_file }}'
    owner: root
    group: root
    mode: '0600'
    setype: etc_t
  notify: restart vsftpd

- name: firewalld is installed
  yum:
    name: firewalld
    state: present

- name: firewalld is started and enabled
  service:
    name: firewalld
    state: started
    enabled: yes

- name: FTP port is open
  firewalld:
    service: ftp
    permanent: true
    state: enabled
    immediate: yes
```

```
- name: Passive FTP data ports allowed through the firewall
firewalld:
  port: 21000-21020/tcp
  permanent: yes
  state: enabled
  immediate: yes
```

- 2.8. Kopieren Sie die Handler aus dem Playbook `ansible-vsftpd.yml` in die Datei `roles/ansible-vsftpd/handlers/main.yml`. Die Datei `roles/ansible-vsftpd/handlers/main.yml` sollte Folgendes enthalten, wenn Sie fertig sind.

```
---
# handlers file for ansible-vsftpd
- name: restart vsftpd
  service:
    name: "{{ vsftpd_service }}"
    state: restarted
```

3. Aktualisieren Sie den Inhalt der Datei `roles/ansible-vsftpd/meta/main.yml`.

Variable	Wert
Autor	Red Hat Training
description	Beispielrolle für RH294
Unternehmen	Red Hat
Lizenz	BSD

- 3.1. Ändern Sie den Wert des Eintrags `author` in `Red Hat Training`.

```
author: Red Hat Training
```

- 3.2. Ändern Sie den Wert des Eintrags `description` in `example role for RH294`.

```
description: example role for RH294
```

- 3.3. Ändern Sie den Wert des Eintrags `company` in `Red Hat`.

```
company: Red Hat
```

- 3.4. Ändern Sie den Wert des Eintrags `license` in `BSD`.

```
license: BSD
```

4. Bearbeiten Sie den Inhalt der Datei `roles/ansible-vsftpd/README.md` für die Rolle, damit die Datei relevante Informationen zur Rolle bereitstellt. Nach der Bearbeitung sollte die Datei Folgendes enthalten:

```
ansible-vsftpd
=====
Example ansible-vsftpd role from Red Hat's "Linux Automation" (RH294)
course.

Role Variables
-----
* defaults/main.yml contains variables used to configure the vsftpd.conf template
* vars/main.yml contains the name of the vsftpd service, the name of the RPM
package, and the location of the service's configuration file

Dependencies
-----
None.

Example Playbook
-----
- hosts: servers
  roles:
    - ansible-vsftpd

License
-----
BSD

Author Information
-----
Red Hat (training@redhat.com)
```

5. Entfernen Sie die nicht verwendeten Verzeichnisse aus der neuen Rolle.

```
[student@workstation review-roles]$ rm -rfv roles/ansible-vsftpd/tests
removed 'roles/ansible-vsftpd/tests/inventory'
removed 'roles/ansible-vsftpd/tests/test.yml'
removed directory: 'roles/ansible-vsftpd/tests'
```

6. Erstellen Sie das neue Playbook vsftpd-configure.yml. Es sollte Folgendes enthalten.

```
---
- name: Install and configure vsftpd
  hosts: ftpservers
  vars:
    vsftpd_anon_root: /mnt/share/
    vsftpd_local_root: /mnt/share/

  roles:
    - ansible-vsftpd
```

```
tasks:  
  - name: /dev/vdb1 is partitioned  
    parted:  
      device: /dev/vdb  
      number: 1  
      label: gpt  
      part_start: 1MiB  
      part_end: 100%  
      state: present  
  
  - name: XFS file system exists on /dev/vdb1  
    filesystem:  
      dev: /dev/vdb1  
      fstype: xfs  
      force: yes  
  
  - name: anon_root mount point exists  
    file:  
      path: '{{ vsftpd_anon_root }}'  
      state: directory  
  
  - name: /dev/vdb1 is mounted on anon_root  
    mount:  
      path: '{{ vsftpd_anon_root }}'  
      src: /dev/vdb1  
      fstype: xfs  
      state: mounted  
      dump: '1'  
      passno: '2'  
    notify: restart vsftpd  
  
  - name: Make sure permissions on mounted fs are correct  
    file:  
      path: '{{ vsftpd_anon_root }}'  
      owner: root  
      group: root  
      mode: '0755'  
      setype: "{{ vsftpd_setype }}"  
      state: directory  
  
  - name: Copy README to the ftp anon_root  
    copy:  
      dest: '{{ vsftpd_anon_root }}/README'  
      content: "Welcome to the FTP server at {{ ansible_fqdn }}\n"  
      setype: '{{ vsftpd_setype }}'
```

7. Ändern Sie das Playbook `site.yml` so, dass statt des Playbooks `ansible-vsftpd.yml` das neu erstellte Playbook `vsftpd-configure.yml` verwendet wird.

```
---  
# FTP Servers playbook  
- import_playbook: vsftpd-configure.yml  
  
# FTP Clients playbook  
- import_playbook: ftpclients.yml
```

8. Verifizieren Sie, dass das Playbook `site.yml` wie gewünscht funktioniert, indem Sie es mit `ansible-playbook` ausführen.

```
[student@workstation review-roles]$ ansible-playbook site.yml
```

## Bewertung

Führen Sie auf `workstation` den Befehl `lab review-roles grade` aus, um den Erfolg dieser Übung zu bestätigen. Beheben Sie sämtliche gemeldeten Fehler, und führen Sie das Skript so lange erneut aus, bis die Durchführung erfolgreich ist.

```
[student@workstation ~]$ lab review-roles grade
```

## Beenden

Führen Sie den Befehl `lab review-roles finish` aus, um die Übungsaufgaben auf `servera` und `serverb` zu löschen.

```
[student@workstation ~]$ lab review-roles finish
```

Hiermit ist die praktische Übung abgeschlossen.

## Anhang A

# Ergänzende Themen

### Ziel

Ermittlung zusätzlicher Themen, die nicht im offiziellen Kurs enthalten sind.

### Abschnitte

- Ermitteln möglicher Ansible-Konfigurationsoptionen

# Ermitteln möglicher Ansible-Konfigurationsoptionen

---

## Ziele

Nach Abschluss dieses Abschnitts sollten die Teilnehmer in der Lage sein, `ansible-config` zu verwenden, um Konfigurationsoptionen zu ermitteln und zu untersuchen sowie festzustellen, welche Optionen gegenüber den Standardeinstellungen geändert wurden.

## Prüfen der Konfigurationsoptionen

Wenn Sie herausfinden möchten, welche Optionen in der Konfigurationsdatei verfügbar sind, verwenden Sie den Befehl `ansible-config list`. Es wird eine vollständige Liste der verfügbaren Konfigurationsoptionen und ihrer Standardeinstellungen angezeigt. Diese Liste kann variieren, abhängig von der installierten Version von Ansible und davon, ob auf Ihrem Kontrollknoten zusätzliche Ansible-Plugins installiert sind.

Jeder durch `ansible-config list` angezeigten Option werden mehrere Schlüsselpaare zugeordnet. Diese Schlüsselpaare geben Auskunft darüber, wie diese Option funktioniert. Zum Beispiel zeigt die Option `ACTION_WARNINGS` die folgenden Schlüsselpaare an:

Schlüssel	Wert	Zweck
<code>description</code>	[Beim Empfang von einer Aufgabenaktion (Modul oder Action-Plugin) gibt Ansible standardmäßig eine Warnung aus. Diese Warnungen können ausgeschaltet werden, indem Sie diese Einstellung auf „False“ setzen.]	Beschreibt, wozu diese Konfigurationsoption dient.
<code>type</code>	<code>boolean</code>	Der Typ für diese Option: <code>boolean</code> bedeutet die Werte „true“ oder „false“.
<code>default</code>	<code>true</code>	Der Standardwert für diese Option.
<code>version_added</code>	2.5	Die Version von Ansible, die diese Option zwecks Abwärtskompatibilität hinzugefügt hat.
<code>ini</code>	{ key: <code>action_warnings</code> , section: <code>defaults</code> }	Welcher Abschnitt derINI-ähnlichen Inventardatei diese Option enthält und der Name dieser Option in der Konfigurationsdatei ( <code>action_warnings</code> im Abschnitt <code>defaults</code> ).

Schlüssel	Wert	Zweck
env	ANSIBLE_ACTION_WARNINGS	Wenn diese Umgebungsvariable festgelegt ist, werden alle in der Konfigurationsdatei vorgenommenen Einstellungen überschrieben.

## Bestimmen der geänderten Konfigurationsoptionen

Wenn Sie mit Konfigurationsdateien arbeiten, möchten Sie möglicherweise herausfinden, welche Optionen andere Werte als die integrierten Standardeinstellungen enthalten.

Sie können dies mit dem Befehl `ansible-config dump -v --only-changed` durchführen. Die Option `-v` zeigt den Speicherort der Datei `ansible.cfg` an, der bei der Verarbeitung des Befehls verwendet wird. Der Befehl `ansible-config` folgt der gleichen Prioritätsfolge, die zuvor für den Befehl `ansible` genannt wurde. Die Ausgabe variiert abhängig vom Speicherort der Datei `ansible.cfg` und davon, aus welchem Verzeichnis der Befehl `ansible-config` ausgeführt wurde.

Im folgenden Beispiel befindet sich eine einzige Ansible-Konfigurationsdatei unter `/etc/ansible/ansible.cfg`. Der Befehl `ansible-config` wird zuerst aus dem Benutzerverzeichnis des Kursteilnehmers ausgeführt, danach aus einem Arbeitsverzeichnis mit den gleichen Ergebnissen:

```
[user@controlnode ~]$ ansible-config dump -v --only-changed
Using /etc/ansible/ansible.cfg as config file
DEFAULT_ROLES_PATH(/etc/ansible/ansible.cfg) = [u'/etc/ansible/roles', u'/usr/share/ansible/roles']

[user@controlnode ~]$ cd /home/student/workingdirectory
[user@controlnode workingdirectory]$ ansible-config dump -v --only-changed
Using /etc/ansible/ansible.cfg as config file
DEFAULT_ROLES_PATH(/etc/ansible/ansible.cfg) = [u'/etc/ansible/roles', u'/usr/share/ansible/roles']
```

Wenn sich jedoch eine benutzerdefinierte `ansible.cfg`-Datei in Ihrem Arbeitsverzeichnis befindet, zeigt derselbe Befehl Informationen basierend darauf an, von wo aus er ausgeführt wird, sowie die entsprechende `ansible.cfg`-Datei.

```
[user@controlnode ~]$ ansible-config dump -v --only-changed
Using /etc/ansible/ansible.cfg as config file
DEFAULT_ROLES_PATH(/etc/ansible/ansible.cfg) = [u'/etc/ansible/roles', u'/usr/share/ansible/roles']

[user@controlnode ~]$ cd /home/student/workingdirectory
[user@controlnode workingdirectory]$ cat ansible.cfg
[defaults]
inventory = ./inventory
remote_user = devops

[user@controlnode workingdirectory]$ ansible-config dump -v --only-changed
```

```
Using /home/student/workingdirectory/ansible.cfg as config file
DEFAULT_HOST_LIST(/home/student/workingdirectory/ansible.cfg) = [u'/home/student/
workingdirectory/inventory']
DEFAULT_REMOTE_USER(/home/student/workingdirectory/ansible.cfg) = devops
```



### Literaturhinweise

Manpage `ansible-config(1)`

### Configuration file: Ansible-Dokumentation

[https://docs.ansible.com/ansible/2.9/installation\\_guide/intro\\_configuration.html](https://docs.ansible.com/ansible/2.9/installation_guide/intro_configuration.html)

## Anhang B

# Ansible Lightbulb-Lizenzierung

# Ansible Lightbulb-Lizenz

---

Teile dieses Kurses basieren auf dem Ansible Lightbulb-Projekt. Das Material aus diesem Projekt ist ab <https://github.com/ansible/lightbulb> mit der folgenden MIT-Lizenz verfügbar.

Copyright 2017 Red Hat, Inc.

Der oben aufgeführte Urheberschutzvermerk und dieser Genehmigungsvermerk sind in alle Kopien oder wesentlichen Teile der Software einzubeziehen.

Hiermit wird jeder Person, die eine Kopie dieser Software und der zugehörigen Dokumentationsdateien (die „Software“) erhält, unentgeltlich die Erlaubnis erteilt, diese Software uneingeschränkt zu benutzen, insbesondere das Recht, Kopien der Software zu nutzen, zu kopieren, zu ändern, zusammenzuführen, zu veröffentlichen, zu verbreiten, unterzulizenziern und/oder zu verkaufen und Personen, welche die Software erhalten, unter den folgenden Bedingungen dieselben Rechte einzuräumen:

Der oben aufgeführte Urheberschutzvermerk und dieser Genehmigungsvermerk sind in alle Kopien oder wesentlichen Teile der Software einzubeziehen.

DIE SOFTWARE WIRD „WIE BESEHEN“ UND OHNE GEWÄHRLEISTUNGEN JEGLICHER ART BEREITGESTELLT, WEDER AUSDRÜCKLICH NOCH KONKLUDENT, EINSCHLIESSLICH, JEDOCH NICHT BESCHRÄNKT AUF GEWÄHRLEISTUNGEN HINSICHTLICH HANDELSÜBLICHKEIT, EIGNUNG FÜR EINEN BESTIMMTEN ZWECK ODER NICHTVERLETZUNG VON RECHTEN DRITTER. UNTER KEINEN UMSTÄNDEN HAFTEN DIE VERFASSER ODER URHEBERRECHTSINHABER FÜR FORDERUNGEN, SCHÄDEN ODER SONSTIGE HAFTUNGSANSPRÜCHE, OB AUFGRUND EINER VERTRAGSVERLETZUNG, EINER UNERLAUBTEN HANDLUNG ODER EINES ANDEREN RECHTSGRUNDSES, DIE SICH AUS ODER IN VERBINDUNG MIT DER SOFTWARE ODER DER VERWENDUNG ODER SONSTIGEN HANDHABUNGEN DER SOFTWARE ERGEBEN.