

Rejoignez les explorateurs, les bâtisseurs et tous ceux qui ont le courage de proposer des solutions nouvelles à des problèmes anciens. Dans le domaine de l'open source, l'innovation dépend entièrement des personnes qui y travaillent.

# RED HAT® TRAINING + CERTIFICATION

## MANUEL D'EXERCICES (ROLE)

Red Hat Ansible Engine 2.7 DO407

**AUTOMATION WITH ANSIBLE**

Édition 1





# AUTOMATION WITH ANSIBLE



# **Red Hat Ansible Engine 2.7 DO407**

## **Automation with Ansible**

**Édition 1 20190110**

**Date de publication 20190110**

Auteurs: Chen Chang, Victor Costea, Dan Kolepp, Artur Glogowski, George Hacker,

Razique Mahroua, Adolfo Vazquez, Snehangshu Karmakar

Éditeur: Steven Bonneville, Seth Kenlon, David O'Brien

Copyright © 2019 Red Hat, Inc.

The contents of this course and all its modules and related materials, including handouts to audience members, are  
Copyright © 2019 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 e-mail [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 Shadowman logo, JBoss, Hibernate, Fedora, the Infinity Logo, and RHCE are trademarks of Red Hat, Inc., registered 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.

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.

Certaines parties de ce cours sont des adaptations du projet Ansible Lightbulb. Le support de ce projet est disponible sur <https://github.com/ansible/lightbulb> sous la licence MIT.

|   |            |
|---|------------|
| <b>Conventions de la documentation</b>  | <b>vii</b> |
| <b>Introduction</b>   | <b>ix</b>  |
| Automation with Ansible .....   | ix         |
| Organisation de l'environnement de formation .....                                  | x          |
| Internationalisation .....  | xiii       |
| <b>1. Présentation d'Ansible</b>  | <b>1</b>   |
| Présentation d'Ansible .....  | 2          |
| Quiz: Présentation d'Ansible .....  | 8          |
| Installation d'Ansible .....  | 10         |
| Exercice guidé: Installation d'Ansible .....  | 15         |
| Résumé .....  | 16         |
| <b>2. Déploiement d'Ansible</b>   | <b>17</b>  |
| Création d'un inventaire Ansible .....  | 18         |
| Exercice guidé: Création d'un inventaire Ansible .....                              | 23         |
| Gestion des fichiers de configuration Ansible .....                                 | 28         |
| Exercice guidé: Gestion des fichiers de configuration Ansible .....                 | 36         |
| Exécution de commandes Ad Hoc .....   | 40         |
| Exercice guidé: Exécution de commandes Ad Hoc .....                                 | 47         |
| Open Lab: Déploiement d'Ansible .....   | 52         |
| Résumé .....  | 61         |
| <b>3. Mise en œuvre de playbooks</b>  | <b>63</b>  |
| Écriture et exécution de playbooks .....  | 64         |
| Exercice guidé: Écriture et exécution de playbooks .....                            | 71         |
| Mise en œuvre de plusieurs plays .....  | 76         |
| Exercice guidé: Mise en œuvre de plusieurs plays .....                              | 86         |
| Open Lab: Mise en œuvre de playbooks .....  | 94         |
| Résumé .....  | 103        |
| <b>4. Gestion des variables et des faits</b>  | <b>105</b> |
| Gestion des variables .....   | 106        |
| Exercice guidé: Gestion des variables .....   | 114        |
| Gérer les secrets .....   | 119        |
| Exercice guidé: Gérer les secrets .....   | 125        |
| Gestion des faits .....   | 128        |
| Exercice guidé: Gestion des faits .....   | 137        |
| Open Lab: Gestion des variables et des faits .....                                  | 142        |
| Résumé .....  | 154        |
| <b>5. Mise en œuvre d'un contrôle de tâche</b>                                      | <b>155</b> |
| Écriture de boucles et de tâches conditionnelles .....                              | 156        |
| Exercice guidé: Écriture de boucles et de tâches conditionnelles .....              | 167        |
| Mise en œuvre des gestionnaires .....   | 176        |
| Exercice guidé: Mise en œuvre des gestionnaires .....                               | 179        |
| Gestion des échecs de tâche .....   | 184        |
| Exercice guidé: Gestion des échecs de tâche .....                                   | 188        |
| Open Lab: Mise en œuvre d'un contrôle de tâche .....                                | 196        |
| Résumé .....  | 205        |
| <b>6. Déploiement de fichiers sur des hôtes gérés</b>                               | <b>207</b> |
| Modification et copie de fichiers sur des hôtes .....                               | 208        |
| Exercice guidé: Modification et copie de fichiers sur des hôtes .....               | 214        |
| Déploiement de fichiers personnalisés avec des modèles Jinja2 .....                 | 223        |
| Exercice guidé: Déploiement de fichiers personnalisés avec des modèles Jinja2 ..... | 228        |
| Open Lab: Déploiement de fichiers sur des hôtes gérés .....                         | 231        |

|  |            |
|--|------------|
| Résumé .....   | 237        |
| <b>7. Gestion de projets volumineux</b>  | <b>239</b> |
| Sélection d'hôtes avec des modèles d'hôte .....                                    | 240        |
| Exercice guidé: Sélection d'hôtes avec des modèles d'hôte .....                    | 249        |
| Gestion des inventaires dynamiques .....   | 256        |
| Exercice guidé: Gestion des inventaires dynamiques .....                           | 262        |
| Configuration d'un parallélisme .....  | 266        |
| Exercice guidé: Configuration d'un parallélisme .....                              | 269        |
| Inclusion et importation de fichiers .....   | 274        |
| Exercice guidé: Inclusion et importation de fichiers .....                         | 279        |
| Open Lab: Gestion de projets volumineux .....                                      | 284        |
| Résumé .....   | 292        |
| <b>8. Simplification des playbooks avec des rôles</b>                              | <b>293</b> |
| Description de la structure d'un rôle .....  | 294        |
| Quiz: Description de la structure d'un rôle .....                                  | 300        |
| Création de rôles .....  | 302        |
| Exercice guidé: Création de rôles .....  | 308        |
| Déploiement de rôles avec Ansible Galaxy .....                                     | 318        |
| Exercice guidé: Déploiement de rôles avec Ansible Galaxy .....                     | 326        |
| Réutilisation du contenu avec des rôles système .....                              | 333        |
| Exercice guidé: Réutilisation du contenu avec des rôles système .....              | 341        |
| Open Lab: Mise en œuvre des rôles .....  | 347        |
| Résumé .....   | 359        |
| <b>9. Résolution des problèmes liés à Ansible</b>                                  | <b>361</b> |
| Résolution de problèmes concernant les playbooks .....                             | 362        |
| Exercice guidé: Résolution de problèmes concernant les playbooks .....             | 365        |
| Résolution des problèmes liés aux hôtes gérés Ansible .....                        | 373        |
| Exercice guidé: Résolution des problèmes liés aux hôtes gérés Ansible .....        | 378        |
| Open Lab: Résolution des problèmes liés à Ansible .....                            | 382        |
| Résumé .....   | 391        |
| <b>10. Automatisation des tâches d'administration Linux</b>                        | <b>393</b> |
| Présentation de l'automatisation des tâches d'administration Linux .....           | 394        |
| Exercice guidé: Gestion des logiciels et des abonnements .....                     | 397        |
| Exercice guidé: Gestion des utilisateurs et de l'authentification .....            | 405        |
| Exercice guidé: Gestion du processus de démarrage et des processus planifiés ..... | 412        |
| Exercice guidé: Gestion du stockage .....  | 422        |
| <b>11. Révision approfondie : Automation with Ansible</b>                          | <b>435</b> |
| Révision complète .....  | 436        |
| Open Lab: Déploiement d'Ansible .....  | 439        |
| Open Lab: Création de playbooks .....  | 445        |
| Open Lab: Création de rôles et utilisation de l'inventaire dynamique .....         | 455        |
| <b>A. Sujets supplémentaires</b>   | <b>469</b> |
| Examen des options de configuration Ansible .....                                  | 470        |
| <b>B. Licences Ansible Lightbulb</b>   | <b>473</b> |
| Licence Ansible Lightbulb .....  | 474        |

# CONVENTIONS DE LA DOCUMENTATION

---



## RÉFÉRENCES

Les « références » indiquent où trouver de la documentation externe se rapportant à un sujet.



## NOTE

Une « remarque » est un conseil, un raccourci ou une approche alternative pour la tâche considérée. Le fait d'ignorer une remarque ne devrait pas entraîner de conséquences négatives, mais vous pourriez passer à côté d'une astuce qui vous simplifierait la vie.



## IMPORTANT

Les cadres « Important » détaillent des éléments qui pourraient aisément être négligés : des changements de configuration qui ne s'appliquent qu'à la session en cours ou des services qui doivent être redémarrés pour qu'une mise à jour soit appliquée. Ignorer un cadre « Important » ne vous fera perdre aucune donnée, mais cela pourrait être source de frustration et d'irritation.



## MISE EN GARDE

Un « avertissement » ne doit pas être ignoré. Le fait d'ignorer un avertissement risque fortement d'entraîner une perte de données.



# INTRODUCTION

## AUTOMATION WITH ANSIBLE

*Automation with Ansible* (DO407) est conçu pour les administrateurs système qui ont besoin d'automatiser le déploiement, la configuration, le déploiement d'applications et l'orchestration.

Les stagiaires apprendront comment installer et configurer Ansible sur un poste de travail de gestion et préparer les hôtes gérés à l'automatisation. Les stagiaires vont également écrire Ansible Playbooks pour automatiser des tâches et les exécuter pour s'assurer que les serveurs sont correctement déployés et configurés. Des exemples d'approches permettant d'automatiser les tâches courantes d'administration de systèmes Linux seront également explorés.

### OBJECTIFS DU COURS

- Automatiser les tâches d'administration de systèmes Linux sur les hôtes gérés avec Ansible.
- Apprendre comment rédiger des playbooks Ansible pour standardiser l'exécution de tâches.
- Gérer le chiffrement pour Ansible avec Ansible Vault.

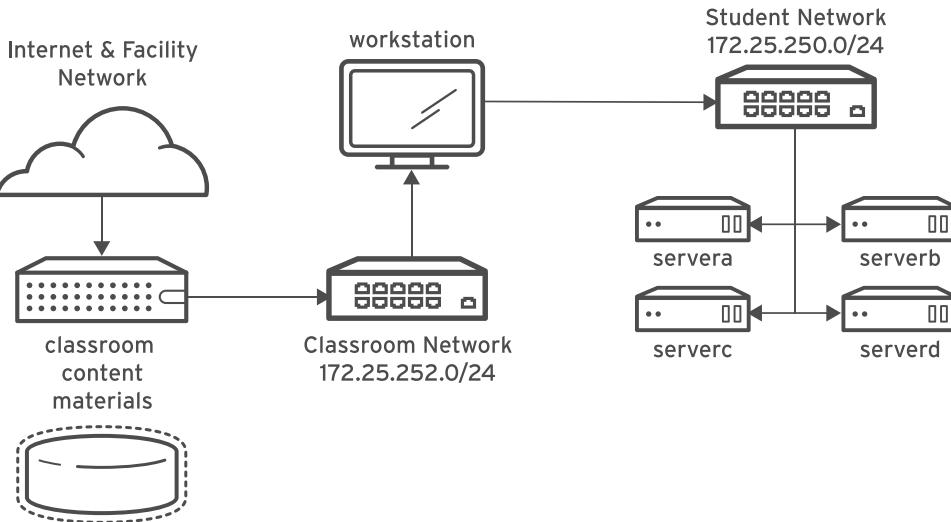
### PUBLIC

- Administrateurs système et du cloud cherchant à automatiser le provisionnement du cloud, la gestion de la configuration, le déploiement d'applications, l'orchestration intraservice et les autres besoins informatiques.

### PRÉREQUIS

- Certification RHCSA (Red Hat Certified System Administrator) dans Red Hat Enterprise Linux) ou expérience équivalente.

# ORGANISATION DE L'ENVIRONNEMENT DE FORMATION



**Figure 0.1: Environnement de formation**

Dans ce cours, le système informatique principal utilisé pour les travaux pratiques est **workstation**. Quatre autres machines sont également utilisées par les stagiaires pour ces activités : **servera**, **serverb**, **serverc** et **serverd**. Ces cinq systèmes se trouvent dans le domaine DNS **lab.example.com**.

Tous les systèmes informatiques des stagiaires possèdent un compte d'utilisateur standard, **student**, protégé par le mot de passe **student**. Le mot de passe **root** est **redhat** sur tous les systèmes des stagiaires.

## Machines de la salle de classe

| NOM DE LA MACHINE           | ADRESSES IP    | RÔLE   |
|-----------------------------|----------------|--|
| workstation.lab.example.com | 172.25.250.254 | Poste de travail graphique utilisé pour exécuter la plupart des commandes de gestion Ansible |
| servera.lab.example.com     | 172.25.250.10  | Hôte géré avec Ansible   |
| serverb.lab.example.com     | 172.25.250.11  | Hôte géré avec Ansible   |
| serverc.lab.example.com     | 172.25.250.12  | Hôte géré avec Ansible   |
| serverd.lab.example.com     | 172.25.250.13  | Hôte géré avec Ansible   |

## Introduction

La machine **workstation** sert également de routeur entre le réseau sur lequel sont connectées les machines des stagiaires et le réseau de la salle de classe. Si le poste de travail **workstation** est arrêté, les autres machines de stagiaires peuvent uniquement accéder aux systèmes qui se trouvent sur le réseau des stagiaires.

Plusieurs systèmes dans la salle de classe proposent des services d'assistance. Deux serveurs, **content.example.com** et **materials.example.com**, hébergent les logiciels et les supports d'atelier utilisés pour les activités pratiques. Les informations relatives à l'utilisation de ces serveurs sont fournies dans les instructions de ces activités.

## Contrôle de votre poste de travail

La partie supérieure de la console indique l'état de votre machine.

### États de la machine

| ÉTAT     | DESCRIPTION  |
|----------|--|
| aucun    | La machine n'a pas encore été démarrée. Une fois démarrée, votre machine est lancée dans un état nouvellement initialisé (le bureau doit être réinitialisé).   |
| starting | La machine est en cours de démarrage.  |
| running  | La machine est en cours d'exécution et disponible (ou, lors du démarrage, le sera bientôt).  |
| stopping | La machine est en cours d'arrêt.   |
| stopped  | La machine est complètement arrêtée. Au démarrage, la machine affiche le même état que lors de son arrêt (le disque est préservé).   |
| impaired | Une connexion réseau à votre machine est impossible. Généralement, cet état apparaît lorsqu'un stagiaire a endommagé les règles de réseau ou de pare-feu. Si le problème persiste après avoir réinitialisé la machine ou s'il se répète, ouvrez un dossier d'assistance. |

Selon l'état de votre machine, une sélection des actions suivantes sera disponible.

### Actions de la machine

| ACTION         | DESCRIPTION  |
|----------------|--|
| Start Station  | Démarre (« sous tension ») la machine.   |
| Stop Station   | Arrête (« hors tension ») la machine, en conservant le contenu de son disque.  |
| Reset Station  | Arrête (« hors tension ») la machine, en réinitialisant l'état antérieur du disque. <b>Attention : tout travail généré sur le disque sera perdu.</b> |
| Refresh        | L'actualisation de la page met à jour l'état de la machine.  |
| Increase Timer | Ajoute 15 minutes à la minuterie pour chaque clic.   |

## Minuterie du poste de travail

Votre inscription à la formation en ligne Red Hat vous donne le droit d'utiliser l'ordinateur pendant un certain temps. Afin de vous aider à conserver le temps qui vous est alloué, les machines ont une minuterie associée réglée sur 60 minutes au démarrage de la machine.

La minuterie fonctionne comme un compte à rebours qui s'écoule lorsque votre machine fonctionne. Si la minuterie arrive à 0, vous pouvez décider d'augmenter sa valeur.

# INTERNATIONALISATION

---

## PRISE EN CHARGE LINGUISTIQUE

Red Hat Enterprise Linux 7 prend officiellement en charge les 22 langues suivantes : allemand, anglais, assamais, bengali, chinois (simplifié), chinois (traditionnel), coréen, espagnol, français, gujarati, hindi, italien, japonais, kannada, malayalam, marathi, odia, portugais (brésilien), pendjabi, russe, tamoul et télougou.

## SÉLECTION DE LA LANGUE PAR UTILISATEUR

Il se peut que les utilisateurs veuillent utiliser, pour leur environnement de bureau, une langue différente de celle utilisée par l'ensemble du système. Il se peut aussi qu'ils veuillent configurer leur compte pour qu'il utilise une autre disposition de clavier ou une autre méthode de saisie.

### Paramètres linguistiques

Dans l'environnement de bureau GNOME, l'utilisateur peut être invité, lors de sa première connexion, à configurer la langue et la méthode de son choix. Dans le cas contraire, la manière la plus simple pour un utilisateur d'ajuster les réglages de langue et de méthode de saisie est d'utiliser l'application Region & Language. Exécutez la commande **gnome-control-center region** ou cliquez sur (User) → Settings dans la barre supérieure. Dans la fenêtre qui s'ouvre, sélectionnez Region & Language. L'utilisateur peut cliquer sur la case Language et sélectionner la langue souhaitée dans la liste qui s'affiche. Cela met également à jour le réglage Formats pour qu'il corresponde aux réglages par défaut pour cette langue. Ces modifications entreront en vigueur à la prochaine connexion de l'utilisateur.

Ces paramètres affectent l'environnement de bureau GNOME et toutes les applications qui y sont lancées, y compris **gnome-terminal**. Toutefois, ils ne s'appliquent pas à ce compte si l'accès a été réalisé via une connexion **ssh** à partir d'un système distant ou d'une console texte locale (ex. : **tty2**).



#### NOTE

L'utilisateur peut faire en sorte que son environnement de shell utilise le même paramètre **LANG** que son environnement graphique, même lorsqu'il se connecte par l'intermédiaire d'une console en mode texte ou par **ssh**. Pour ce faire, on peut placer le code suivant ou son équivalent dans le fichier **~/.bashrc** de l'utilisateur. Ce code fourni en exemple règle la langue utilisée pour une connexion en mode texte pour qu'elle corresponde à celle configurée pour l'environnement de bureau GNOME :

```
i=$(grep 'Language=' /var/lib/AccountService/users/${USER} \
| sed 's/Language=//')
if [ "$i" != "" ]; then
    export LANG=$i
fi
```

Le japonais, le coréen, le chinois et d'autres langues à jeu de caractères autre que le latin peuvent ne pas s'afficher correctement sur les consoles locales en mode texte.

## Introduction

Il est possible d'obliger chaque commande à utiliser une autre langue, en réglant la variable **LANG** depuis la ligne de commande :

```
[user@host ~]$ LANG=fr_FR.utf8 date  
jeu. avril 24 17:55:01 CDT 2014
```

Les commandes suivantes continuent d'utiliser la langue par défaut du système. La commande **locale** peut être utilisée pour vérifier la valeur actuelle de **LANG**, ainsi que d'autres variables d'environnement connexes.

## Paramètres de la méthode de saisie

GNOME 3 de Red Hat Enterprise Linux 7 utilise automatiquement le système de sélection de méthode de saisie IBus qui permet de changer facilement et rapidement la disposition du clavier et les méthodes de saisie.

L'application Region & Language peut aussi servir à activer d'autres méthodes de saisie. Dans la fenêtre de l'application Region & Language, le cadre Input Sources présente les méthodes de saisie actuellement disponibles. Par défaut, English (US) peut être la seule méthode disponible. Sélectionnez English (US), puis cliquez sur l'icône du clavier pour afficher la disposition actuelle du clavier.

Pour ajouter une nouvelle méthode de saisie, cliquez sur le bouton + dans le coin inférieur gauche de la fenêtre Input Sources. Une fenêtre Add an Input Source s'ouvre. Sélectionnez votre langue, puis la méthode de saisie ou la disposition de clavier souhaitée.

Après avoir configuré plusieurs méthodes de saisie, l'utilisateur peut passer rapidement de l'une à l'autre en saisissant **Super+Space** (parfois appelé **Windows+Space**). Par ailleurs, un *indicateur d'état* s'affiche dans la barre supérieure de l'environnement GNOME. Celui-ci a deux fonctions : il indique la méthode de saisie active et joue le rôle de menu vous permettant de passer d'une méthode de saisie à l'autre ou de sélectionner les fonctions avancées de méthodes de saisie plus complexes.

Certaines des méthodes sont marquées par des engrenages, qui indiquent qu'elles ont des options de configuration et des possibilités avancées. Par exemple, la méthode de saisie japonaise Japonais (Kana Kanji) permet à l'utilisateur de préparer un texte en caractères latins et d'utiliser les touches **Flèche vers le bas** et **Flèche vers le haut** pour sélectionner les caractères à utiliser.

Les anglophones américains peuvent également trouver cette méthode utile. Pour English (United States) par exemple, la disposition de clavier est English (international avec touches mortes en AltGr), qui considère la touche **AltGr** (ou la touche **Alt**) de droite sur un clavier PC à 104-105 touches comme une touche de modification « Maj secondaire » et comme touche d'activation des touches mortes pour la saisie des caractères supplémentaires. Le Dvorak et d'autres dispositions sont également proposées.

**NOTE**

Si l'utilisateur connaît le code Unicode d'un caractère, il peut le saisir dans l'environnement de bureau GNOME en combinant les touches **Ctrl+Maj+U** suivies du code correspondant au caractère. Après avoir appuyé sur les touches **Ctrl+Maj+U**, un **u** souligné s'affiche pour indiquer que le système attend la saisie du code du caractère.

Par exemple, le code de caractère de la lettre minuscule grecque lambda est U +03BB et peut être saisi en appuyant sur les touches **Ctrl+Maj+U**, puis **03bb**, puis **Entrée**.

## PARAMÈTRES LINGUISTIQUES PAR DÉFAUT POUR L'ENSEMBLE DU SYSTÈME

La langue du système est réglée par défaut sur Anglais US. Elle fait appel l'encodage Unicode UTF-8 (**en\_US.utf8**) pour son jeu de caractères, mais celui-ci peut être modifié lors de l'installation ou plus tard.

Depuis la ligne de commande, l'utilisateur *root* peut modifier les paramètres linguistiques à l'échelle du système à l'aide de la commande **localectl**. Si **localectl** est exécuté sans argument, il affiche les paramètres linguistiques à l'échelle du système.

Pour définir une langue au niveau du système, exécutez la commande **localectl set-locale LANG=locale**, où *locale* est la valeur **\$LANG** correspondante décrite dans le tableau « Language Codes Reference ». Les changements seront pris en compte lors de la prochaine connexion de l'utilisateur et seront stockés dans le fichier **/etc/locale.conf**.

```
[root@host ~]# localectl set-locale LANG=fr_FR.utf8
```

Dans GNOME, les administrateurs peuvent modifier ce paramètre dans Region & Language en cliquant sur le bouton Login Screen dans le coin supérieur droit de la fenêtre. La modification de la langue (Language) de l'écran de connexion ajustera également le paramètre linguistique pour l'ensemble du système, stocké dans le fichier de configuration **/etc/locale.conf**.

**IMPORTANT**

Les consoles texte locales telles que **tty2** disposent d'un nombre inférieur de polices que les sessions **gnome-terminal** et **ssh**. Par exemple, les caractères japonais, coréens et chinois peuvent ne pas s'afficher correctement dans une console texte locale. Pour cette raison, il est plus logique de choisir l'anglais ou une autre langue à jeu de caractères latins pour ce type de console en mode texte.

De même, les consoles locales en mode texte reconnaissent moins de méthodes de saisie. Ce point est géré séparément de l'environnement graphique du bureau. On peut configurer les paramètres de saisie globaux par l'intermédiaire de **localectl**, à la fois pour les consoles locales virtuelles en mode texte et pour l'environnement graphique X11. Consultez les pages de manuel **localectl(1)**, **kbd(4)** et **vconsole.conf(5)** pour plus d'informations.

## MODULES LINGUISTIQUES

Quand vous utilisez des langues autres que l'anglais, vous devez installer des « packs de langues » supplémentaires pour fournir d'autres traductions, dictionnaires, etc. Pour afficher la liste des packs de langues disponibles, exéutez la commande **yum langavailable**. Pour afficher la liste des packs de langues actuellement installés sur le système, exéutez **yum langlist**. Pour ajouter un pack de langues supplémentaire, lancez **yum langinstall code**, où *code* correspond au code entre crochets situé après le nom de la langue dans la sortie de **yum langavailable**.



### RÉFÉRENCES

Pages de manuel **locale(7)**, **localectl(1)**, **kbd(4)**, **locale.conf(5)**, **vconsole.conf(5)**, **unicode(7)**, **utf-8(7)** et **yum-langpacks(8)**

Les conversions entre le nom des présentations X11 de l'environnement graphique de bureau et leur nom dans **localectl** se trouvent dans le fichier **/usr/share/X11/xkb/rules/base.lst**.

## RÉFÉRENCE DES CODES DE LANGUE

### Codes de langue

| LANGUE                 | VALEUR \$LANG |
|------------------------|---------------|
| Anglais (États-Unis)   | en_US.utf8    |
| Assamais               | as_IN.utf8    |
| Bengali                | bn_IN.utf8    |
| Chinois (simplifié)    | zh_CN.utf8    |
| Chinois (traditionnel) | zh_TW.utf8    |
| Français               | fr_FR.utf8    |
| Allemand               | de_DE.utf8    |
| Gujarati               | gu_IN.utf8    |
| Hindi                  | hi_IN.utf8    |
| Italien                | it_IT.utf8    |
| Japonais               | ja_JP.utf8    |
| Kannada                | kn_IN.utf8    |
| Coréen                 | ko_KR.utf8    |
| Malayalam              | ml_IN.utf8    |
| Marathi                | mr_IN.utf8    |

| <b>LANGUE</b>         | <b>VALEUR \$LANG</b> |
|-----------------------|----------------------|
| Odia                  | or_IN.utf8           |
| Portugais (brésilien) | pt_BR.utf8           |
| Pendjabi              | pa_IN.utf8           |
| Russe                 | ru_RU.utf8           |
| Espagnol              | es_ES.utf8           |
| Tamoul                | ta_IN.utf8           |
| Télougou              | te_IN.utf8           |



## CHAPITRE 1

# PRÉSENTATION D'ANSIBLE

### PROJET

Décrire les concepts Ansible et installer Red Hat Ansible Engine.

### OBJECTIFS

- Décrire les concepts, l'architecture et des cas d'utilisation d'Ansible.
- Installer Ansible sur un nœud de contrôle et décrire la distinction entre communauté Ansible et Red Hat Ansible Engine.

### SECTIONS

- Présentation d'Ansible (avec quiz)
- Installation d'Ansible (et exercice guidé)

# PRÉSENTATION D'ANSIBLE

## OBJECTIF

À la fin de cette section, les stagiaires doivent pouvoir décrire les concepts d'Ansible, son architecture et les cas d'utilisation courants.

## QU'EST-CE QU'ANSIBLE ?

Ansible est une plateforme d'automatisation Open Source. Il s'agit d'un *langage simple d'automatisation* qui peut parfaitement décrire une infrastructure d'application informatique dans les playbooks Ansible. C'est aussi un *moteur d'automatisation* qui exécute des playbooks Ansible.

Ansible peut gérer des tâches puissantes d'automatisation et s'adapter à de nombreux workflows et environnements différents. De plus, les nouveaux utilisateurs d'Ansible peuvent devenir rapidement productifs, car ce langage est simple à utiliser.

### Ansible est simple

Les playbooks Ansible fournissent une automatisation lisible par l'utilisateur. Cela signifie que les playbooks sont des outils d'automatisation faciles à lire, à comprendre et à modifier. Les utilisateurs n'ont besoin d'aucune compétence particulière en codage pour les écrire. Les playbooks exécutent des tâches dans l'ordre. La simplicité de la conception des playbooks les rend utilisables par toutes les équipes, ce qui permet aux personnes novices dans Ansible d'être rapidement productives.

### Ansible est puissant

Vous pouvez utiliser Ansible pour le déploiement d'applications, la gestion de la configuration ainsi que l'automatisation du workflow et du réseau. Ansible vous permet d'orchestrer le cycle de vie complet d'une application.

### Ansible n'utilise pas d'agent

Ansible est construit sur une architecture sans agent. En général, Ansible se connecte aux hôtes qu'il gère en utilisant OpenSSH ou WinRM et exécute des tâches, la plupart du temps (mais pas systématiquement) en transmettant par push de petits programmes, appelés *modules Ansible*, à ces hôtes. Ces programmes servent à placer le système dans un état spécifique. Tout module transmis par push est supprimé lorsqu'Ansible a terminé ses tâches. Vous pouvez commencer à utiliser Ansible quasiment immédiatement, car aucun agent n'exige d'approbation pour être utilisé et déployé sur les hôtes gérés. En l'absence d'agent ou d'autre infrastructure de sécurité personnalisée, Ansible s'avère plus efficace et sécurisé que d'autres solutions.

Ansible présente de nombreux atouts :

- *Prise en charge multiplateforme* : Ansible offre une prise en charge sans agent de Linux, Windows, UNIX et des périphériques réseau dans des environnements physiques, virtuels, de cloud et de conteneurs.
- *Automatisation lisible par l'utilisateur* : les playbooks Ansible, écrits sous la forme de fichiers texte YAML, offrent une facilité de lecture et la garantie que tout le monde comprenne les actions effectuées.

- *Description parfaite des applications* : chaque modification peut être effectuée par des playbooks Ansible et chaque aspect de l'environnement d'application peut être décrit et documenté.
- *Gestion facile grâce au contrôle de versions* : les playbooks et les projets Ansible sont en texte brut. Ils peuvent être traités comme du code source et placés dans le système existant de contrôle de versions.
- *Prise en charge d'inventaires dynamiques* : la liste de machines gérées par Ansible peut être mise à jour de manière dynamique à partir de sources externes. Ainsi, la liste correcte et actuelle de tous les serveurs gérés est constamment capturée, indépendamment de l'infrastructure ou de l'emplacement.
- *Intégration facile de l'orchestration avec d'autres systèmes* : HP SA, Puppet, Jenkins, Red Hat Satellite et d'autres systèmes existants dans l'environnement peuvent être exploités et intégrés dans votre workflow Ansible.

## ANSIBLE : LE LANGAGE DES DEVOPS



Figure 1.1: Ansible tout au long du cycle de vie des applications

La communication est essentielle pour les DevOps. Ansible est le premier langage d'automatisation qui peut être lu et écrit dans le domaine informatique. Il est également le seul moteur qui automatisé le cycle de vie des applications et le pipeline de livraison continue, du début jusqu'à la fin.

## CONCEPTS ET ARCHITECTURE ANSIBLE

L'architecture Ansible propose deux types de machine : les *nœuds de contrôle* et les *hôtes gérés*. Ansible est installé et exécuté à partir d'un nœud de contrôle, et cette machine contient également des copies de vos fichiers de projet Ansible. Un nœud de contrôle peut être l'ordinateur portable d'un administrateur, un système partagé par plusieurs administrateurs ou un serveur exécutant Red Hat Ansible Tower.

Les hôtes gérés sont listés dans un *inventaire*, qui organise également ces systèmes en groupes afin de faciliter la gestion collective. Vous pouvez définir l'inventaire dans un fichier texte statique ou le déterminer de manière dynamique au moyen de scripts qui obtiennent des informations provenant de sources externes.

Au lieu d'écrire des scripts complexes, les utilisateurs Ansible créent des *plays* de haut niveau pour s'assurer qu'un hôte ou un groupe d'hôtes soit dans un état particulier. Un play effectue une série de *tâches* sur les hôtes, dans l'ordre spécifié par le play. Ces plays sont exprimés au format YAML dans un fichier texte. Un fichier contenant un ou plusieurs plays est appelé un *playbook*.

Chaque tâche exécute un *module*, qui est une petite section de code (écrite en Python, PowerShell ou un autre langage) avec des arguments spécifiques. Chaque module constitue un outil de votre boîte à outils. Ansible est fourni avec des centaines de modules utiles qui peuvent accomplir une grande variété de tâches d'automatisation. Ils peuvent agir sur les fichiers système, installer des logiciels ou effectuer des appels d'API.

Lorsqu'un module est utilisé dans une tâche, il permet généralement de s'assurer qu'un aspect particulier concernant la machine est dans un état particulier. Par exemple, une tâche utilisant un module peut garantir qu'un fichier existe et dispose d'autorisations et de contenus particuliers, tandis qu'une tâche utilisant un module différent peut garantir qu'un système de fichiers particuliers est monté. Si le système ne se trouve pas à l'état souhaité, c'est la tâche qui le met à l'état. À l'inverse, si le système est déjà au bon état, il ne fait rien. Si une tâche échoue, le comportement par défaut d'Ansible consiste à interrompre le reste du playbook pour les hôtes qui ont subi un échec.

Les tâches, les plays et les playbooks sont conçus pour être *idempotents*. Cela signifie que vous pouvez exécuter en toute sécurité un playbook sur les mêmes hôtes plusieurs fois. Lorsque vos systèmes sont dans le bon état, le playbook n'effectue aucune modification lorsque vous l'exéutez. Cela signifie que vous devez pouvoir exécuter un playbook sur les mêmes hôtes plusieurs fois en toute sécurité. Lorsque vos systèmes sont dans le bon état, le playbook ne devrait effectuer aucune modification lorsque vous l'exéutez. Il existe une poignée de modules que vous pouvez utiliser pour exécuter des commandes arbitraires, mais vous devez utiliser ces modules avec soin pour vous assurer qu'ils s'exécutent de manière idempotente.

Ansible utilise également des *plug-ins*. Les plug-ins sont du code que vous pouvez ajouter à Ansible pour l'étendre et l'adapter aux nouvelles utilisations et plateformes.

L'architecture d'Ansible ne repose pas sur l'utilisation d'agent. Généralement, lorsqu'un administrateur exécute un playbook Ansible ou une commande ad hoc, le nœud de contrôle se connecte à l'hôte géré à l'aide de SSH (par défaut) ou WinRM. Cela signifie que les clients n'ont pas besoin d'avoir un agent spécifique Ansible installé sur les hôtes gérés, ni d'autoriser le trafic réseau spécial vers un port non standard.

Red Hat Ansible Tower est une structure d'entreprise qui vous permet de contrôler, sécuriser et gérer l'automatisation Ansible à l'échelle. Vous pouvez l'utiliser, entre autres, pour contrôler les utilisateurs qui ont accès aux playbooks ainsi que les hôtes auxquels ils accèdent, pour partager l'utilisation des informations d'identification SSH sans permettre aux utilisateurs de transférer ou d'afficher le contenu, ainsi que pour vous connecter à toutes vos tâches Ansible et gérer l'inventaire. Il fournit une interface utilisateur Web et une API RESTful. Il ne s'agit pas d'une partie centrale d'Ansible, mais d'un produit distinct qui vous aide à utiliser Ansible plus efficacement avec une équipe ou à grande échelle.

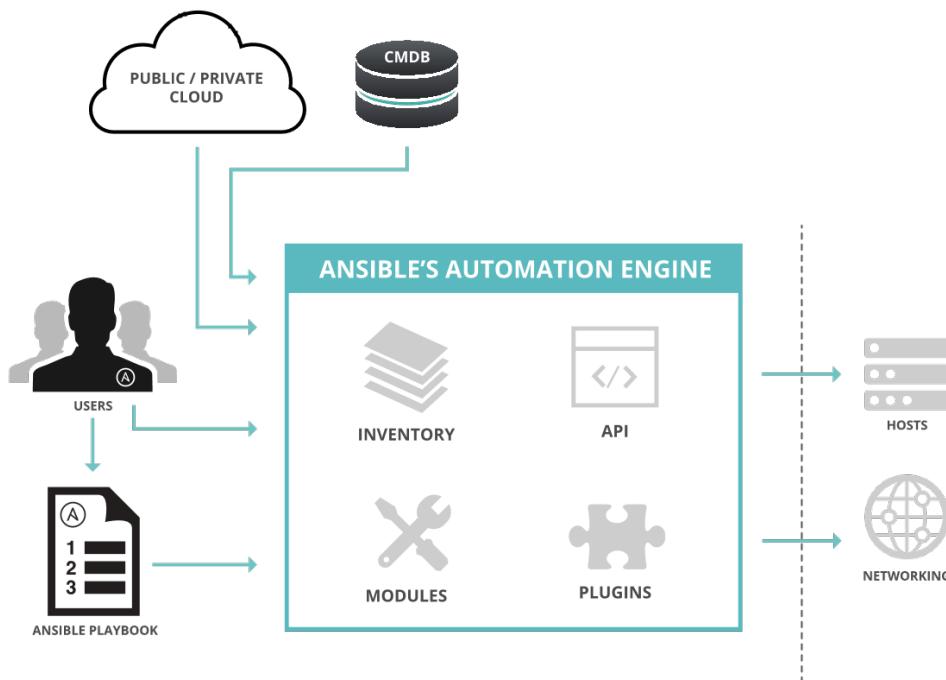


Figure 1.2: Architecture Ansible

## L'ART D'ANSIBLE

### La complexité tue la productivité

La simplicité paie. De par sa conception, Ansible est doté d'outils simples à utiliser. Par conséquent, l'automatisation est facile à écrire et à lire. Il peut vous permettre de simplifier la façon dont vous créez l'automatisation.

### Meilleure lisibilité

Le langage d'automatisation Ansible repose sur des fichiers texte simples, déclaratifs et faciles à lire pour les utilisateurs. Correctement écrits, les playbooks Ansible peuvent clairement documenter l'automatisation du workflow.

### Langage déclaratif

Ansible est un moteur qui permet d'appliquer l'état souhaité. Il cerne le problème de l'automatisation des déploiements informatiques en exprimant ces derniers en terme d'état à appliquer aux systèmes. L'objectif d'Ansible consiste à placer les systèmes dans l'état souhaité, en apportant uniquement les modifications nécessaires. Traiter Ansible comme un langage de script n'est pas la meilleure approche.

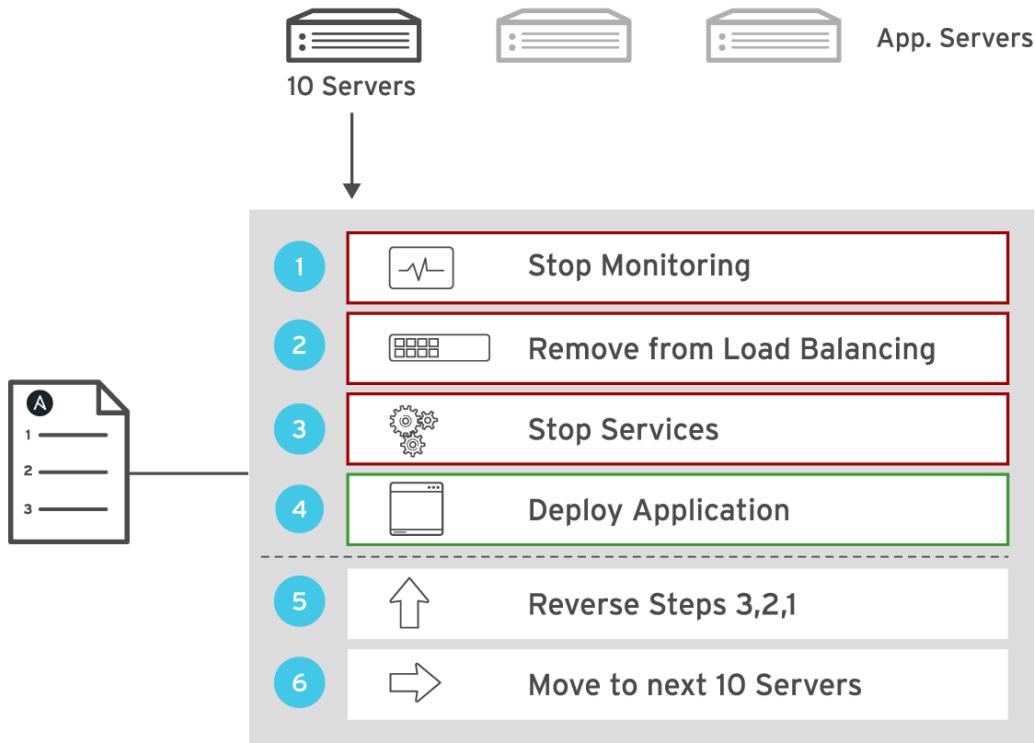


Figure 1.3: Ansible offre une automatisation complète

## EXEMPLES D'UTILISATION

Contrairement à d'autres outils, Ansible associe et réunit l'orchestration et la gestion de configuration, le provisioning et le déploiement d'applications dans une seule plateforme facile à utiliser.

Parmi les cas d'utilisation d'Ansible, nous pouvons citer :

### Gestion de la configuration

De nombreux utilisateurs expérimentés découvrent pour la première fois la plateforme d'automatisation Ansible au travers du cas d'utilisation courant d'Ansible qui porte sur la centralisation de la gestion et du déploiement des fichiers de configuration.

### Le déploiement d'applications

Lorsque vous définissez votre application avec Ansible et gérez le déploiement avec Red Hat Ansible Tower, les équipes peuvent traiter efficacement l'ensemble du cycle de vie de l'application, du développement à la production.

### Déploiement

Vous devez déployer ou installer les applications sur les systèmes. Ansible et Red Hat Ansible Tower peuvent contribuer à rationaliser le processus des systèmes de déploiement, que vous soyez au stade de l'amorçage PXE et de l'installation automatisée des serveurs ou des machines virtuelles, ou à celui de la création des machines virtuelles ou des instances cloud à partir de modèles. Vous devez déployer ou installer les applications sur les systèmes.

### Livraison continue

La création d'un pipeline CI/CD impose la coordination et l'adhésion de nombreuses équipes. Dans ce contexte, la plateforme d'automatisation simple est l'élément essentiel qui peut être

utilisé par tous les membres de votre organisation. Avec les playbooks Ansible, les applications sont correctement déployées (et gérées) tout au long de leur cycle de vie.

#### Sécurité et conformité

Lorsque votre politique de sécurité est définie dans les playbooks Ansible, l'analyse et l'application des règles de sécurité à l'échelle du site peuvent être intégrées dans d'autres processus automatisés. Au lieu d'être considérés en aval, ces aspects sont intégrés dans l'ensemble des déploiements.

#### Orchestration

Les configurations seules ne définissent pas votre environnement. Vous devez définir comment plusieurs configurations interagissent et garantir que les éléments hétérogènes puissent être gérés dans leur ensemble.



#### RÉFÉRENCES

##### **Ansible**

<https://www.ansible.com>

##### **Fonctionnement d'Ansible**

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

## ► QUIZ

# PRÉSENTATION D'ANSIBLE

Répondez aux questions suivantes en sélectionnant une réponse :

► 1. Quel terme décrit le mieux l'architecture Ansible ?

- a. Sans agent
- b. Client/Serveur
- c. Axé sur les événements
- d. Sans état

► 2. Quel protocole réseau Ansible utilise-t-il, par défaut, pour communiquer avec des nœuds gérés ?

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

► 3. Quel fichier définit les actions effectuées par Ansible sur les nœuds gérés ?

- a. Inventaire d'hôtes
- b. Manifeste
- c. Playbook
- d. Script

► 4. Quelle syntaxe est utilisée pour définir les playbooks Ansible ?

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

## ► SOLUTION

# PRÉSENTATION D'ANSIBLE

Répondez aux questions suivantes en sélectionnant une réponse :

► 1. Quel terme décrit le mieux l'architecture Ansible ?

- a. Sans agent
- b. Client/Serveur
- c. Axé sur les événements
- d. Sans état

► 2. Quel protocole réseau Ansible utilise-t-il, par défaut, pour communiquer avec des nœuds gérés ?

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

► 3. Quel fichier définit les actions effectuées par Ansible sur les nœuds gérés ?

- a. Inventaire d'hôtes
- b. Manifeste
- c. Playbook
- d. Script

► 4. Quelle syntaxe est utilisée pour définir les playbooks Ansible ?

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

# INSTALLATION D'ANSIBLE

---

## OBJECTIFS

Au terme de cette section, les stagiaires doivent pouvoir installer Ansible sur un nœud de contrôle et décrire la distinction entre la communauté Ansible et Red Hat Ansible Engine.

**Figure 1.3: Installation d'Ansible**

## ANSIBLE OU RED HAT ANSIBLE ENGINE ?

Red Hat fournit des logiciels Ansible dans des canaux de distribution spéciaux, ce qui est pratique pour les abonnés Red Hat Enterprise Linux, et vous pouvez utiliser ces progiciels normalement.

Cependant, si vous souhaitez une prise en charge formelle pour Ansible et ses modules, Red Hat propose un abonnement spécial à cette fin, Red Hat Ansible Engine. Cela ajoute une prise en charge technique formelle avec des contrats de niveau de service et une étendue de couverture publiée pour Ansible et ses modules de base. Plus d'informations sur l'étendue de cette prise en charge sont disponibles sur *Red Hat Ansible Engine Life Cycle* [<https://access.redhat.com/support/policy/updates/ansible-engine>].

## NŒUDS DE CONTRÔLE

Ansible est simple à installer. Le logiciel Ansible doit uniquement être installé sur le ou les nœuds de contrôle à partir desquels Ansible est exécuté. Les hôtes gérés par Ansible n'exigent pas qu'Ansible soit installé sur eux. Cette installation implique un nombre relativement limité d'étapes et présente des besoins minimes.

Le nœud de contrôle doit être un système Linux ou UNIX. Microsoft Windows n'est pas pris en charge en tant que nœud de contrôle, bien que les systèmes Windows puissent être des hôtes gérés.

Vous devez installer Python 2 (version 2.7 ou ultérieure) ou Python 3 (version 3.5 ou ultérieure) sur le nœud de contrôle. Pour voir si une version de Python appropriée est installée sur un système Red Hat Enterprise Linux, utilisez la commande **yum**.

```
[root@controlnode ~]# yum list installed python
Loaded plugins: langpacks, search-disabled-repos
Installed Packages
python.x86_64      2.7.5-48.el7      installed
```

Des informations sur la façon d'installer le progiciel *ansible* sur un système Red Hat Enterprise Linux est disponible dans l'article de la base de connaissances *Comment télécharger et installer Red Hat Ansible Engine ?* [<https://access.redhat.com/articles/3174981>].

Ansible est en développement rapide en amont, et donc Red Hat Ansible Engine a un cycle de vie rapide. Plus d'informations sur le cycle de vie actuel sont disponibles sur <https://access.redhat.com/support/policy/updates/ansible-engine>.

Red Hat fournit actuellement plusieurs canaux de distribution pour Red Hat Enterprise Linux 7 Server, comme indiqué dans le tableau suivant.

| NOM DU CANAL DE DISTRIBUTION                | DESCRIPTION   |
|---|---|
| <code>rhel-7-server-ansible-2-rpms</code>   | Dernière mise à jour de la version principale de Red Hat Ansible Engine 2 pour RHEL 7. La version mineure peut changer dans ce canal. |
| <code>rhel-7-server-ansible-2.7-rpms</code> | Red Hat Ansible Engine 2.7 pour RHEL 7.   |
| <code>rhel-7-server-ansible-2.6-rpms</code> | Red Hat Ansible Engine 2.6 pour RHEL 7.   |
| <code>rhel-7-server-ansible-2.5-rpms</code> | Red Hat Ansible Engine 2.5 pour RHEL 7.   |
| <code>rhel-7-server-ansible-2.4-rpms</code> | Red Hat Ansible Engine 2.4 pour RHEL 7.   |

Si vous disposez d'abonnements Red Hat Enterprise Linux, vous pouvez utiliser ces canaux de distribution pour installer Ansible avec une assistance limitée. Si vous avez besoin d'une assistance Ansible plus complète, vous pouvez acheter des abonnements intégraux à Red Hat Ansible Engine et les associer à vos systèmes avant d'activer les canaux de distribution, comme indiqué dans l'article de la base de connaissances.

Si vous disposez d'un abonnement Red Hat Ansible Engine, la procédure d'installation pour Red Hat Ansible Engine 2 est la suivante :

1. Enregistrez votre système sur Red Hat Subscription Manager.

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

2. Attachez votre abonnement Red Hat Ansible Engine. Cette commande vous aide à trouver votre abonnement Red Hat Ansible Engine :

```
[root@host ~]# subscription-manager list --available
```

3. Utilisez l'ID de pool de l'abonnement pour attacher le pool au système.

```
[root@host ~]# subscription-manager attach --pool=<engine-subscription-pool>
```

4. Activez le référentiel Red Hat Ansible Engine.

```
[root@host ~]# subscription-manager repos \
> --enable rhel-7-server-ansible-2-rpms
```

5. Installez Red Hat Ansible Engine.

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

Si vous utilisez la version avec une assistance limitée fournie avec votre abonnement Red Hat Enterprise Linux, utilisez la procédure suivante :

1. Activez le référentiel Red Hat Ansible Engine.

```
[root@host ~]# subscription-manager refresh
[root@host ~]# subscription-manager repos \
```

```
> --enable rhel-7-server-ansible-2-rpms
```

2. Installez Red Hat Ansible Engine.

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



### IMPORTANT

Si vous utilisez déjà Ansible mais que vous l'avez installé depuis le canal de distribution Red Hat Enterprise Linux 7 Extras, sachez qu'Ansible et ses dépendances ne seront plus mises à jour via le canal Extras. Les canaux de distribution officiels abordés dans cette section remplacent cette méthode de distribution.

Pour plus d'informations, consultez l'article de la base de connaissances *Ansible obsolète dans le canal Extras* [<https://access.redhat.com/articles/3359651>].

Les nœuds de contrôle Ansible doivent communiquer avec les hôtes gérés sur le réseau. SSH est utilisé par défaut. Toutefois, d'autres protocoles peuvent s'avérer nécessaires si des périphériques réseau ou des systèmes Microsoft Windows sont gérés. Sur les nœuds de contrôle Red Hat Enterprise Linux, si vous gérez des systèmes Microsoft Windows, vous devez disposer de la version 0.3.0 ou ultérieure du paquetage RPM *python2-winrm* installé (qui fournit le paquetage Python *pywinrm*).

## HÔTES GÉRÉS

L'un des avantages des hôtes gérés, c'est qu'ils n'ont pas besoin qu'un agent Ansible spécial soit installé. Le nœud de contrôle Ansible se connecte aux hôtes gérés à l'aide d'un protocole réseau standard afin de garantir que les systèmes sont dans l'état spécifié.

Les hôtes gérés peuvent présenter certaines exigences en fonction du mode de connexion du nœud de contrôle et des modules qui s'y exécutent.

Les hôtes gérés Linux et UNIX doivent être dotés de Python 2 (version 2.6 ou ultérieure) ou Python 3 (version 3.5 ou ultérieure) pour que la plupart des modules puissent fonctionner. Pour Red Hat Enterprise Linux, installez le paquetage *python*.

Si SELinux est activé sur les hôtes gérés, vous devez également installer le paquetage *libselinux-python* avant d'utiliser les modules qui sont liés aux fonctions de copie, de fichier ou de modèle. (Notez que si les autres composants Python sont installés, vous pouvez utiliser les modules Ansible yum ou package pour vous assurer que ce paquetage est également installé.)

Certains modules peuvent avoir leurs propres exigences supplémentaires. Par exemple, le module *dnf*, qui peut servir à installer des paquetages sur les systèmes Fedora actuels, a besoin du paquetage *python-dnf*.

**NOTE**

Certains modules n'ont pas besoin de Python. Par exemple, les arguments transmis au module `raw` sont exécutés directement via le shell distant configuré plutôt que via le sous-système du module. Cela peut être utile pour gérer les périphériques qui ne disposent pas de Python ou sur lesquels il est impossible d'installer ce dernier, ou encore pour amorcer Python sur un système qui en est dépourvu.

Toutefois, le module `raw` est difficile à utiliser de manière sûre et idempotente. Si vous pouvez utiliser un module normal à la place, il est généralement préférable d'éviter d'utiliser `raw` et les modules de commande similaires. Ce point sera abordé ultérieurement dans le cours.

## Hôtes gérés Microsoft Windows

Ansible inclut un certain nombre de modules spécialement conçus pour les systèmes Microsoft Windows. Ceux-ci sont listés dans la section Modules Windows [[https://docs.ansible.com/ansible/2.7/modules/list\\_of\\_windows\\_modules.html](https://docs.ansible.com/ansible/2.7/modules/list_of_windows_modules.html)] de l'index des modules Ansible.

La plupart des modules spécialement conçus pour les hôtes gérés Microsoft Windows ont besoin que PowerShell 3.0 ou ultérieur soit installé sur l'hôte géré au lieu de Python. En outre, vous devez configurer une connexion à distance PowerShell sur les hôtes gérés. Ansible nécessite également au moins l'installation de .NET Framework 4.0 ou version ultérieure sur des hôtes gérés Windows.

Les exemples de ce cours utilisent des hôtes gérés Linux. Les différences spécifiques et les réglages nécessaires lors de la gestion des hôtes gérés Microsoft Windows ne sont pas abordés en détail. Vous trouverez un complément d'informations sur le site Web d'Ansible à l'adresse [https://docs.ansible.com/ansible/2.7/user\\_guide/windows.html](https://docs.ansible.com/ansible/2.7/user_guide/windows.html).

## Périphériques réseau gérés

Vous pouvez également utiliser l'automatisation Ansible pour configurer des périphériques réseau gérés tels que les routeurs et les commutateurs. Ansible comprend un grand nombre de modules spécialement conçus à cet effet. Cela inclut la prise en charge des périphériques réseau Cisco IOS, IOS XR et NX-OS ; Juniper Junos ; Arista EOS ; et VyOS, entre autres.

Vous pouvez écrire des playbooks Ansible pour des périphériques réseau en utilisant les mêmes techniques de base que celles que vous utilisez lorsque vous écrivez des playbooks pour des serveurs. Comme la plupart des périphériques réseau ne peuvent pas exécuter Python, Ansible exécute les modules réseau sur le nœud de contrôle, et non sur les hôtes gérés. Des méthodes de connexion spéciales sont également utilisées pour communiquer avec les périphériques réseau, généralement à l'aide de l'ILC sur SSH, de XML sur SSH ou de l'API sur HTTP(S).

Ce cours ne couvre pas en détail l'automatisation de la gestion des périphériques réseau. Pour plus d'informations sur ce sujet, consultez *Ansible for Network Automation* [<https://docs.ansible.com/ansible/latest/network/index.html>] sur le site Web de la communauté Ansible, ou suivez notre cours alternatif *Ansible for Network Automation* (DO457) [<https://www.redhat.com/en/services/training/do457-ansible-network-automation>].



## RÉFÉRENCES

Page de manuel (1)ansible

**Top Support Policies for Red Hat Ansible Automation**

<https://access.redhat.com/ansible-top-support-policies>

**Installation Guide – Ansible Documentation**

[http://docs.ansible.com/ansible/2.7/installation\\_guide/intro\\_installation.html](http://docs.ansible.com/ansible/2.7/installation_guide/intro_installation.html)

**Windows Guides – Ansible Documentation**

[https://docs.ansible.com/ansible/2.7/user\\_guide/windows.html](https://docs.ansible.com/ansible/2.7/user_guide/windows.html)

**Ansible for Networking Automation – Ansible Documentation**

<https://docs.ansible.com/ansible/2.7/network/index.html>

## ► EXERCICE GUIDÉ

# INSTALLATION D'ANSIBLE

Dans cet exercice, vous allez installer Ansible sur un nœud de contrôle exécutant Red Hat Entreprise Linux.

## RÉSULTAT

Vous devez pouvoir installer Ansible sur un nœud de contrôle.

Connectez-vous à `workstation` en tant qu'utilisateur `student` avec le mot de passe `student` et exécutez `lab intro-install setup`. Ce script de configuration s'assure que l'hôte géré, `servera`, est accessible sur le réseau.

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

- 1. Vérifiez que Python 2.7 est installé sur `workstation`.

```
[student@workstation ~]$ yum list installed python
```

- 2. Installez Ansible sur `workstation` pour qu'il serve de nœud de contrôle.

```
[student@workstation ~]$ sudo yum install ansible
```

- 3. Vérifiez que l'installation d'Ansible est réussie. Exécutez la commande `ansible` avec l'option `--version`.

```
[student@workstation ~]$ ansible --version
ansible 2.7.1
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/home/student/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/site-packages/ansible
  executable location = /usr/bin/ansible
  python version = 2.7.5 (default, Sep 12 2018, 05:31:16) [GCC 4.8.5 20150623 (Red
  Hat 4.8.5-36)]
```

## Nettoyage

Sur `workstation`, exécutez le script `lab intro-install cleanup` pour effacer l'exercice.

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

L'exercice guidé est maintenant terminé.

# RÉSUMÉ

---

Dans ce chapitre, vous avez appris les principes suivants :

- Ansible est une plateforme d'automatisation Open Source qui peut s'adapter à de nombreux flux de travail et environnements.
- Ansible sert à gérer différents types de systèmes, y compris les serveurs exécutant Linux, Microsoft Windows ou UNIX et les périphériques réseau.
- Les playbooks Ansible sont des fichiers texte lisibles par l'utilisateur qui décrivent l'état souhaité d'une infrastructure informatique.
- Ansible repose sur une architecture sans agent, il est installé sur un nœud de contrôle. De plus, les clients n'ont pas besoin de logiciel d'agent spécial.
- Ansible se connecte aux hôtes gérés à l'aide de protocoles réseau standard tels que SSH et exécute du code ou des commandes sur les hôtes gérés pour garantir qu'ils sont dans l'état qu'il a spécifié.

## CHAPITRE 2

# DÉPLOIEMENT D'ANSIBLE

### PROJET

Configurer Ansible pour gérer des hôtes et exécuter les commandes Ansible ad hoc.

### OBJECTIFS

- Décrire les concepts d'inventaire Ansible et gérer un fichier d'inventaire statique.
- Décrire où se trouvent les fichiers de configuration Ansible, comment Ansible les sélectionne, et les modifier pour appliquer les modifications aux paramètres par défaut.
- Exécuter une seule tâche d'automatisation Ansible à l'aide d'une commande ad hoc et expliquer certains cas d'utilisation de commandes ad hoc.

### SECTIONS

- Création d'un inventaire Ansible (et exercice guidé)
- Gestion des fichiers de configuration Ansible (et exercice guidé)
- Exécution de commandes Ad Hoc (et exercice guidé)

### ATELIER

- Déploiement d'Ansible

# CRÉATION D'UN INVENTAIRE ANSIBLE

---

## OBJECTIF

Au terme de cette section, les stagiaires doivent pouvoir décrire les concepts d'inventaire Ansible et gérer un fichier inventaire statique.

**Figure 2.0: Création d'un inventaire Ansible**

## L'INVENTAIRE

Un *inventaire* définit une collection d'hôtes qu'Ansible va gérer. Ces hôtes peuvent également être affectés à des *groupes*, qui peuvent être gérés collectivement. Les groupes peuvent contenir des groupes enfants et les hôtes peuvent être membres de plusieurs groupes. L'inventaire peut également définir des variables qui s'appliquent aux hôtes et aux groupes qu'il définit.

Les inventaires d'hôtes peuvent être définis de deux manières différentes. Un inventaire d'hôte *statique* peut être défini par un fichier texte. Un inventaire d'hôte *dynamique* peut être généré par un script ou un autre programme selon les besoins, en utilisant des fournisseurs d'informations externes.

## INVENTAIRE STATIQUE

Un fichier d'inventaire statique est un fichier texte qui spécifie les hôtes gérés qu'Ansible cible. Vous pouvez écrire ce fichier en utilisant un certain nombre de formats différents, y compris un format de type INI et un format exprimé sous forme de document YAML. Le format de type INI est très courant et sera utilisé pour la plupart des exemples de ce cours.

Dans sa forme la plus simple, un fichier d'inventaire de type INI est une liste de noms d'hôtes ou d'adresses IP d'hôtes gérés, chacun sur une seule ligne :

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

Normalement, cependant, vous organisez les hôtes gérés en *groupes d'hôtes*. Les groupes d'hôtes vous permettent d'exécuter Ansible plus efficacement sur la base d'une collection de systèmes. Dans ce cas, chaque section commence par un nom de groupe d'hôtes qui figure entre crochets ([]). Vient ensuite le nom d'hôte ou une adresse IP pour chaque hôte géré du groupe, chaque élément sur une seule ligne.

Dans l'exemple suivant, l'inventaire d'hôtes définit deux groupes d'hôtes : `webservers` et `db-servers`.

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

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

Les hôtes peuvent figurer dans plusieurs groupes. En fait, il est recommandé d'organiser vos hôtes en plusieurs groupes, éventuellement organisés de différentes manières en fonction du rôle de l'hôte, de son emplacement physique, du fait qu'il soit en production ou non, et ainsi de suite. Cela vous permet d'appliquer plus facilement des plays Ansible à des hôtes spécifiques.

```
[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
```



### IMPORTANT

Deux groupes d'hôtes existent toujours :

- Le groupe d'hôtes `all` contient tous les hôtes listés explicitement dans l'inventaire.
- Le groupe d'hôtes `ungrouped` contient tous les hôtes listés explicitement dans l'inventaire qui ne sont membres d'aucun autre groupe.

## Définition de groupes imbriqués

Les inventaires d'hôtes Ansible peuvent inclure des groupes de groupes d'hôtes. Pour cela, il faut créer un nom de groupe d'hôtes avec le suffixe `:children`. L'exemple suivant crée un nouveau groupe appelé `north-america`, qui comprend tous les hôtes des groupes `usa` et `canada`.

```
[usa]
```

```
washington1.example.com
washington2.example.com

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

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

Un groupe peut avoir à la fois des hôtes gérés et des groupes enfants en tant que membres. Par exemple, dans l'inventaire précédent, vous pourriez ajouter une section **[north-america]** qui a sa propre liste d'hôtes gérés. Cette liste d'hôtes serait fusionnée avec les hôtes supplémentaires dont le groupe **north-america** hérite de ses groupes enfants.

## Simplification des spécifications d'hôte avec des plages

Vous pouvez spécifier des plages dans les noms d'hôte ou les adresses IP pour simplifier les inventaires d'hôtes Ansible. Vous pouvez spécifier des plages numériques ou alphabétiques. La syntaxe des plages se présente comme suit :

```
[START:END]
```

Les plages correspondent à toutes les valeurs comprises entre les clauses *START* et *END* incluses. Par exemple :

- 192.168.[4:7].[0:255] correspond à toutes les adresses IPv4 sur le réseau 192.168.4.0/22 (de 192.168.4.0 à 192.168.7.255).
- server[01:20].example.com correspond à tous les hôtes nommés server01.example.com jusqu'à server20.example.com.
- [a:c].dns.example.com correspond aux hôtes nommés a.dns.example.com, b.dns.example.com et c.dns.example.com.
- 2001:db8::[a:f] correspond à toutes les adresses IPv6 de 2001:db8::a jusqu'à 2001:db8::f.

Si des plages numériques commencent par des zéros, ces derniers sont inclus dans l'expression. Le deuxième exemple ci-dessus ne correspond pas à **server1.example.com** mais bien à **server07.example.com**. Pour illustrer cela, l'exemple suivant utilise des plages pour simplifier les définitions des groupes **[usa]** et **[canada]** de l'exemple précédent :

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

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

## Vérification de l'inventaire

En cas de doute, utilisez la commande **ansible** pour tester la présence de la machine dans l'inventaire :

```
[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):

```

Vous pouvez exécuter la commande suivante pour lister tous les hôtes d'un groupe :

```

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

```



### IMPORTANT

Si l'inventaire contient un hôte et un groupe d'hôtes portant le même nom, la commande **ansible** imprime un avertissement et cible l'hôte. Le groupe d'hôtes n'est pas pris en compte.

Il existe plusieurs manières de gérer cette situation, la plus simple étant de s'assurer que les groupes d'hôtes n'utilisent pas les mêmes noms que les hôtes de l'inventaire.

## Remplacer l'emplacement de l'inventaire

Le fichier **/etc/ansible/hosts** est considéré comme le fichier d'inventaire statique par défaut du système. Toutefois, la pratique normale consiste à ne pas utiliser ce fichier mais à définir un emplacement différent pour les fichiers d'inventaire dans votre fichier de configuration Ansible. Ceci est couvert dans la section suivante.

Les commandes **ansible** et **ansible-playbook** que vous utilisez pour exécuter des commandes ad hoc et des playbooks plus tard dans le cours peuvent également spécifier l'emplacement d'un fichier d'inventaire sur la ligne de commande avec l'option **--inventory PATHNAME** ou **-i PATHNAME**, où **PATHNAME** correspond au chemin d'accès au fichier d'inventaire souhaité.

## Définition de variables dans l'inventaire

Les valeurs des variables utilisées dans les playbooks peuvent être spécifiées dans les fichiers d'inventaire d'hôtes. Ces variables s'appliquent uniquement à des hôtes ou à des groupes d'hôtes spécifiques. Normalement, il est préférable de définir ces *variables d'inventaire* dans des répertoires spéciaux, et non directement dans le fichier d'inventaire. Ce point sera abordé plus en détail ultérieurement.

## INVENTAIRE DYNAMIQUE

Vous pouvez aussi générer dynamiquement des informations d'inventaire Ansible en utilisant les informations fournies par des bases de données externes. La communauté Open Source a écrit un certain nombre de scripts d'inventaire dynamique disponibles à partir du projet Ansible en amont. Si ces scripts ne répondent pas à vos besoins, vous pouvez également écrire les vôtres.

Par exemple, un programme d'inventaire dynamique peut contacter votre serveur Red Hat Satellite ou votre compte Amazon EC2 et utiliser les informations stockées pour créer un inventaire Ansible. Comme le programme le fait lorsque vous exécutez Ansible, il peut remplir l'inventaire avec les

informations mises à jour fournies par le service à mesure que de nouveaux hôtes sont ajoutés et que les anciens sont supprimés.

Ce point sera abordé plus en détail ultérieurement dans le cours.



## RÉFÉRENCES

### Inventaire : Documentation Ansible

[http://docs.ansible.com/ansible/2.7/user\\_guide/intro\\_inventory.html](http://docs.ansible.com/ansible/2.7/user_guide/intro_inventory.html)

## ► EXERCICE GUIDÉ

# CRÉATION D'UN INVENTAIRE ANSIBLE

Dans cet exercice, vous allez créer un nouvel inventaire statique contenant des hôtes et des groupes.

## RÉSULTATS

Vous devez pouvoir créer des inventaires statiques par défaut et personnalisés.

Connectez-vous en tant qu'utilisateur student sur workstation, puis exécutez **lab deploy-inventory setup**. Ce script de configuration vérifie que les hôtes gérés, servera, serverb, serverc, serverd, sont accessibles sur le réseau.

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

- ▶ 1. Modifiez **/etc/ansible/hosts** pour inclure servera.lab.example.com en tant qu'hôte géré.
  - 1.1. Ajoutez servera.lab.example.com à la fin du fichier d'inventaire par défaut, **/etc/ansible/hosts**.

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

servera.lab.example.com
```

- 1.2. Continuez à modifier le fichier d'inventaire **/etc/ansible/hosts** en ajoutant un groupe **[webservers]** au bas du fichier avec le serveur serverb.lab.example.com en tant que membre du groupe.

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

servera.lab.example.com

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

- ▶ 2. Vérifiez les hôtes gérés dans le fichier d'inventaire **/etc/ansible/hosts**.
  - 2.1. Utilisez la commande **ansible all --list-hosts** pour lister tous les hôtes gérés du fichier d'inventaire par défaut.

```
[student@workstation ~]$ ansible all --list-hosts
```

```
hosts (2):
servera.lab.example.com
serverb.lab.example.com
```

- 2.2. Utilisez la commande **ansible ungrouped --list-hosts** pour lister uniquement les hôtes gérés n'appartenant pas à un groupe.

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

- 2.3. Utilisez la commande **ansible webservers --list-hosts** pour lister uniquement les hôtes gérés n'appartenant pas au groupe webservers.

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

- ▶ 3. Créez un fichier d'inventaire statique personnalisé nommé **inventory** dans le répertoire de travail **/home/student/deploy-inventory**.

Les informations sur vos quatre hôtes gérés sont listées dans le tableau suivant. Vous allez affecter chaque hôte à plusieurs groupes à des fins de gestion en fonction de l'objectif de l'hôte, de la ville où il se trouve et de l'environnement de déploiement auquel il appartient.

En outre, les groupes pour les villes américaines (Raleigh et Mountain View) doivent être configurés comme enfants du groupe us afin que les hôtes aux États-Unis puissent être gérés en tant que groupe.

#### Spécifications d'inventaire du serveur

| NOM D'HÔTE              | OBJET       | EMPLACEMENT   | ENVIRONNEMENT |
|-------------------------|-------------|---------------|---------------|
| servera.lab.example.com | Serveur Web | Raleigh       | Développement |
| serverb.lab.example.com | Serveur Web | Raleigh       | Testing       |
| serverc.lab.example.com | Serveur Web | Mountain View | Production    |

| NOM D'HÔTE              | OBJET       | EMPLACEMENT | ENVIRONNEMENT |
|-------------------------|-------------|-------------|---------------|
| serverd.lab.example.com | Serveur Web | Londres     | Production    |

- 3.1. Créez le répertoire de travail **/home/student/deploy-inventory**.

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

- 3.2. Créez un fichier **inventory** dans le répertoire de travail **/home/student/deploy-inventory**. Utilisez le tableau Spécifications d'inventaire du serveur comme guide. Modifiez le fichier **inventory** et ajoutez le contenu suivant :

```
[student@workstation ~]$ cd /home/student/deploy-inventory
[student@workstation deploy-inventory]$ vim inventory
[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. Utilisez des variations de la commande **ansible host-or-group -i inventory --list-hosts** pour vérifier les hôtes et les groupes gérés dans le fichier d'inventaire **/home/student/deploy-inventory/inventory** personnalisé.



### IMPORTANT

Votre commande **ansible** doit inclure l'option **-i inventory**. Cela permet à **ansible** d'utiliser votre fichier **inventory** dans le répertoire de travail en cours au lieu du fichier d'inventaire **/etc/ansible/hosts** du système.

- 4.1. Utilisez la commande **ansible all -i inventory --list-hosts** pour lister tous les hôtes gérés.

```
[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. Utilisez la commande **ansible ungrouped -i inventory --list-hosts** pour lister tous les hôtes gérés listés dans le fichier d'inventaire mais ne faisant pas partie d'un groupe. Ce fichier d'inventaire ne contient aucun hôte géré non groupé.

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

hosts (0):
```

- 4.3. Utilisez la commande **ansible development -i inventory --list-hosts** pour lister tous les hôtes gérés listés dans le groupe **development**.

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

- 4.4. Utilisez la commande **ansible testing -i inventory --list-hosts** pour lister tous les hôtes gérés listés dans le groupe **testing**.

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

- 4.5. Utilisez la commande **ansible production -i inventory --list-hosts** pour lister tous les hôtes gérés listés dans le groupe **production**.

```
[student@workstation deploy-inventory]$ ansible production -i inventory \
```

```
> --list-hosts
hosts (2):
serverc.lab.example.com
serverd.lab.example.com
```

- 4.6. Utilisez la commande **ansible us -i inventory --list-hosts** pour lister tous les hôtes gérés listés dans le groupe us.

```
[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. Nous vous encourageons à expérimenter d'autres variantes pour confirmer les entrées d'hôtes gérés dans le fichier d'inventaire personnalisé.

## Nettoyage

À partir de workstation, exécutez le script **lab deploy-inventory cleanup** pour effacer cet exercice.

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

L'exercice guidé est maintenant terminé.

# GESTION DES FICHIERS DE CONFIGURATION ANSIBLE

---

## OBJECTIFS

Au terme de cette section, les stagiaires doivent pouvoir décrire où se trouvent les fichiers de configuration Ansible, comment ils sont sélectionnés par Ansible, et les modifier pour appliquer les modifications aux paramètres par défaut.

**Figure 2.0: Gestion des fichiers de configuration Ansible**

## CONFIGURATION D'ANSIBLE

Le comportement d'une installation Ansible peut être personnalisé en modifiant les paramètres figurant dans le fichier de configuration Ansible. Ansible choisit son fichier de configuration parmi plusieurs emplacements possibles sur le nœud de contrôle.

### Utilisation de `/etc/ansible/ansible.cfg`

Le paquetage `ansible` fournit un fichier de configuration de base sous `/etc/ansible/ansible.cfg`. Ce fichier est utilisé si aucun autre fichier de configuration n'est trouvé.

### En utilisant `~/.ansible.cfg`

Ansible recherche un fichier `.ansible.cfg` dans le répertoire personnel de l'utilisateur. Cette configuration est utilisée au lieu de `/etc/ansible/ansible.cfg`, si elle existe et s'il n'y a pas de fichier `ansible.cfg` dans le répertoire de travail actuel.

### Utilisation de `./ansible.cfg`

Si un fichier `ansible.cfg` existe dans le répertoire où est exécutée la commande `ansible`, il est utilisé à la place du fichier global ou du fichier personnel de l'utilisateur. Cela permet aux administrateurs de créer une structure de répertoire dans laquelle les différents environnements ou projets sont hébergés dans des répertoires distincts, chaque répertoire contenant un fichier de configuration personnalisé avec un ensemble unique de paramètres.



#### IMPORTANT

La pratique recommandée est de créer un fichier `ansible.cfg` dans un répertoire à partir duquel vous exécutez les commandes Ansible. Ce répertoire contient également n'importe quel fichier utilisé par votre projet Ansible, comme un inventaire et un playbook par exemple. C'est l'emplacement le plus couramment utilisé pour le fichier de configuration Ansible. En pratique, il est inhabituel d'utiliser un fichier `~/.ansible.cfg` ou `/etc/ansible/ansible.cfg`.

### Utilisation de la variable d'environnement `ANSIBLE_CONFIG`

Vous pouvez utiliser différents fichiers de configuration en les plaçant dans des répertoires distincts, puis en exécutant les commandes Ansible à partir du répertoire approprié, mais cette méthode peut s'avérer restrictive et difficile à gérer quand le nombre de fichiers de configuration augmente. Une option plus souple consiste à définir l'emplacement du fichier de configuration à

L'aide de la variable d'environnement **ANSIBLE\_CONFIG**. Quand cette variable est définie, Ansible utilise le fichier de configuration spécifié par la variable plutôt que n'importe lequel des fichiers de configuration mentionnés précédemment.

## ORDRE DE PRIORITÉ DES FICHIERS DE CONFIGURATION

L'ordre de recherche d'un fichier de configuration est inversé par rapport à la liste précédente. Le premier fichier situé dans l'ordre de recherche est celui sélectionné par Ansible. Ansible utilise uniquement les paramètres de configuration du premier fichier trouvé.

Tout fichier spécifié par la variable d'environnement **ANSIBLE\_CONFIG** va remplacer tous les autres fichiers de configuration. Si cette variable n'est pas définie, le système vérifie ensuite la présence d'un fichier **ansible.cfg** dans le répertoire dans lequel la commande **ansible** a été exécutée. S'il ne s'y trouve pas, le système recherche le fichier **.ansible.cfg** dans le répertoire personnel de l'utilisateur. Le fichier **/etc/ansible/ansible.cfg** global est utilisé si aucun autre fichier de configuration n'est trouvé.

En raison de la multitude d'emplacements dans lesquels peuvent être placés les fichiers de configuration Ansible, il peut s'avérer difficile de déterminer quel fichier de configuration est utilisé par Ansible. Vous pouvez exécuter la commande **ansible --version** pour identifier clairement la version d'Ansible installée et le fichier de configuration utilisé.

```
[user@controlnode ~]$ ansible --version
ansible 2.7.0
  config file = /etc/ansible/ansible.cfg
...output omitted...
```

Un autre moyen d'afficher le fichier de configuration Ansible en cours consiste à utiliser l'option **-v** lors de l'exécution des commandes Ansible en ligne de commande.

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

Ansible utilise uniquement les paramètres du fichier de configuration avec la priorité la plus élevée. Même s'il existe d'autres fichiers avec des ordres de priorité inférieurs, leurs paramètres sont ignorés et non associés à ceux présents dans le fichier de configuration sélectionné. Par conséquent, si vous choisissez de créer votre propre fichier de configuration au lieu du fichier de configuration **/etc/ansible/ansible.cfg** global, vous devez dupliquer tous les paramètres souhaités de ce fichier dans votre propre fichier de configuration au niveau utilisateur. Les paramètres non définis dans le fichier de configuration au niveau de l'utilisateur restent définis sur les paramètres par défaut intégrés, même s'ils sont définis sur une autre valeur par le fichier de configuration global.

## GESTION DES PARAMÈTRES DANS LE FICHIER DE CONFIGURATION

Le fichier de configuration Ansible se compose de plusieurs sections dont chacune contient des paramètres définis en tant que paires clé-valeur. Les titres de section sont affichés entre crochets. Pour un fonctionnement de base, utilisez les deux sections suivantes :

- **[defaults]** définit les valeurs par défaut pour le fonctionnement d'Ansible

- **[privilege\_escalation]** configure la manière dont Ansible effectue l'élévation des priviléges sur les hôtes gérés

Par exemple, voici un fichier **ansible.cfg** typique :

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

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

Les directives de ce fichier sont expliquées dans le tableau suivant :

#### Configuration Ansible

| DIRECTIVE              | DESCRIPTION  |
|------------------------|--|
| <b>inventory</b>       | Spécifie le chemin d'accès au fichier d'inventaire.  |
| <b>remote_user</b>     | Nom de l'utilisateur à utiliser pour se connecter sur les hôtes gérés. S'il n'est pas spécifié, le nom de l'utilisateur actuel est utilisé.                          |
| <b>ask_pass</b>        | Indique s'il faut spécifier ou non un mot de passe SSH. Peut être <b>false</b> si vous utilisez l'authentification par clé publique SSH.                             |
| <b>become</b>          | Indique s'il faut changer automatiquement d'utilisateur sur l'hôte géré (généralement à <b>root</b> ) après la connexion. Cela peut aussi être spécifié par un play. |
| <b>become_method</b>   | Comment changer d'utilisateur (généralement <b>sudo</b> , qui est la valeur par défaut, mais <b>su</b> est une possibilité).   |
| <b>become_user</b>     | L'utilisateur vers lequel basculer sur l'hôte géré (généralement <b>root</b> , qui est la valeur par défaut).  |
| <b>become_ask_pass</b> | Indique s'il faut demander ou non un mot de passe pour votre <b>become_method</b> . La valeur par défaut est <b>false</b> .  |

## CONFIGURATION DES CONNEXIONS

Ansible doit savoir comment communiquer avec ses hôtes gérés. L'une des raisons les plus courantes liées à la modification du fichier de configuration vise à contrôler les méthodes et les utilisateurs qu'Ansible utilise pour administrer les hôtes gérés. Voici certaines des informations nécessaires :

- L'emplacement de l'inventaire qui liste les hôtes gérés et les groupes d'hôtes

- Le protocole de connexion à utiliser pour communiquer avec les hôtes gérés (par défaut, SSH), et si un port réseau non standard est nécessaire ou non pour se connecter au serveur
- L'utilisateur distant à utiliser sur les hôtes gérés ; il peut s'agir de `root` ou d'un utilisateur non privilégié
- Si l'utilisateur distant ne dispose pas de privilèges, Ansible doit savoir s'il doit essayer d'augmenter les privilèges à `root` et comment procéder (par exemple, en utilisant `sudo`)
- S'il faut ou non demander un mot de passe SSH ou un mot de passe `sudo` pour se connecter ou obtenir des privilèges

## Emplacement de l'inventaire

Dans la section **[defaults]**, la directive **inventory** peut pointer directement vers un fichier d'inventaire statique ou vers un répertoire contenant plusieurs fichiers d'inventaire statiques et/ou des scripts d'inventaire dynamique.

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

## Paramètres de connexion

Par défaut, Ansible se connecte aux hôtes gérés à l'aide du protocole SSH. Les paramètres les plus importants qui commandent la façon dont Ansible se connecte aux hôtes gérés sont définis dans la section **[defaults]**.

Par défaut, Ansible tente de se connecter à l'hôte géré en utilisant le même nom d'utilisateur que l'utilisateur local exécutant les commandes Ansible. Pour spécifier un autre utilisateur distant, définissez le paramètre **remote\_user** sur ce nom d'utilisateur.

Si l'utilisateur local exécutant Ansible a des clés privées SSH configurées qui lui permettent de s'authentifier en tant qu'utilisateur distant sur les hôtes gérés, Ansible se connecte automatiquement. Si ce n'est pas le cas, vous pouvez configurer Ansible pour demander à l'utilisateur local le mot de passe utilisé par l'utilisateur distant en définissant la directive **ask\_pass = true**.

```
[defaults]
inventory = ./inventory

remote_user = root
ask_pass = true
```

En supposant que vous utilisez un nœud de contrôle Linux et OpenSSH sur vos hôtes gérés, si vous pouvez vous connecter en tant qu'utilisateur distant avec un mot de passe, alors vous pouvez probablement configurer l'authentification par clé SSH, ce qui vous permettrait de définir **ask\_pass = false**.

La première étape consiste à s'assurer que l'utilisateur sur le nœud de contrôle a une paire de clés SSH configurée dans `~/.ssh`. Pour cela, vous pouvez exécuter la commande **ssh-keygen**.

Pour un seul hôte géré existant, vous pouvez installer votre clé publique sur l'hôte géré et utiliser la commande **ssh-copy-id** pour remplir votre fichier `~/.ssh/known_hosts` local avec sa clé d'hôte, comme suit :

```
[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.
```



### NOTE

Vous pouvez également utiliser Ansible Playbook pour déployer votre clé publique sur le compte `remote_user` sur *tous* les hôtes gérés à l'aide du module `authorized_key`.

Ce cours n'a pas encore décrit les playbooks Ansible en détail. Un play qui permet de s'assurer que votre clé publique est déployée sur les comptes `root` des hôtes gérés peut se présenter comme suit :

```
- 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
```

Étant donné que l'hôte géré ne dispose pas encore de l'authentification par clé SSH configurée, vous devez exécuter le playbook à l'aide de la commande `ansible-playbook` avec l'option `--ask-pass` pour que la commande s'authentifie en tant qu'utilisateur distant.

## Privilege Escalation (augmentation des privilèges)

Pour des raisons de sécurité et d'audit, Ansible peut avoir besoin de se connecter à des hôtes distants en tant qu'utilisateur non privilégié avant d'augmenter les privilèges pour obtenir un accès administrateur en tant que `root`. Cela peut être configuré dans la section **[privilege\_escalation]** du fichier de configuration Ansible.

Pour activer l'augmentation des privilèges par défaut, définissez la directive `become = true` dans le fichier de configuration. Même si cette option est définie par défaut, il existe plusieurs

façons de l'ignorer lors de l'exécution de commandes ad hoc ou de playbooks Ansible. (Par exemple, vous pouvez vouloir exécuter une tâche ou un play sans augmenter les privilèges.)

La directive **become\_method** spécifie comment augmenter les privilèges. Plusieurs options sont disponibles, mais la valeur par défaut consiste à utiliser **sudo**. De même, la directive **become\_user** spécifie l'utilisateur vers lequel passer, mais la valeur par défaut est **root**.

Si le mécanisme **become\_method** choisi requiert que l'utilisateur entre un mot de passe pour augmenter les privilèges, vous pouvez définir la directive **become\_ask\_pass = true** dans le fichier de configuration.



### NOTE

Sur Red Hat Enterprise Linux 7, la configuration par défaut de **/etc/sudoers** permet à tous les utilisateurs du groupe **wheel** d'utiliser **sudo** pour devenir **root** après avoir entré leur mot de passe.

Pour permettre à un utilisateur (**someuser** dans l'exemple suivant) d'utiliser **sudo** pour devenir **root** sans mot de passe, vous pouvez, par exemple, installer un fichier avec les directives appropriées dans le répertoire **/etc/sudoers.d** (possédé par **root**, avec les permissions octales 0400) :

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

Pensez aux implications en matière de sécurité de l'approche que vous choisissez pour l'augmentation des privilèges. Organisations et déploiements différents peuvent être confrontés à divers compromis.

L'exemple de fichier **ansible.cfg** suivant suppose que vous pouvez vous connecter aux hôtes gérés en tant que **someuser** en utilisant l'authentification par clé SSH, et que **someuser** peut utiliser **sudo** pour exécuter les commandes en tant que **root** sans entrer de mot de passe :

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

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

Le tableau suivant résume certaines des directives les plus fréquemment modifiées dans le fichier de configuration Ansible.

### Paramètres Ansible

| PARAMÈTRE        | DESCRIPTION                          |
|------------------|--------------------------------------|
| <b>inventory</b> | Emplacement de l'inventaire Ansible. |

| PARAMÈTRE              | DESCRIPTION   |
|------------------------|---|
| <b>remote_user</b>     | Compte utilisateur distant utilisé pour établir des connexions aux hôtes gérés.                         |
| <b>ask_pass</b>        | Demander un mot de passe à utiliser lors de la connexion en tant qu'utilisateur distant.                |
| <b>become</b>          | Activer ou désactiver l'augmentation des privilèges pour les opérations effectuées sur les hôtes gérés. |
| <b>become_method</b>   | Méthode d'augmentation des privilèges à utiliser sur les hôtes gérés.                                   |
| <b>become_user</b>     | Compte utilisateur vers lequel augmenter les privilèges sur les hôtes gérés.                            |
| <b>become_ask_pass</b> | Indique si un mot de passe doit être demandé pour l'augmentation des privilèges sur les hôtes gérés.    |

## Connexions non-SSH

Le protocole utilisé par Ansible pour se connecter aux hôtes gérés est défini par défaut sur **smart**, ce qui détermine le moyen le plus efficace d'utiliser SSH. Cela peut être réglé sur d'autres valeurs de plusieurs façons.

Par exemple, il existe une exception à la règle selon laquelle SSH est utilisé par défaut. Si vous n'avez pas de **localhost** dans votre inventaire, Ansible met en place une entrée *implicit localhost* pour vous permettre d'exécuter des commandes et des playbooks ad hoc qui ciblent **localhost**. Cette entrée d'inventaire spécial n'est pas incluse dans les groupes d'hôtes **all** ou **ungrouped**. En outre, au lieu d'utiliser le type de connexion SSH **smart**, Ansible se connecte à celui-ci à l'aide du type de connexion **local** spécial par défaut.

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

hosts (1):
  localhost
```

Le type de connexion **local** ignore le paramètre **remote\_user** et exécute les commandes directement sur le système local. Si l'élévation de privilèges est utilisée, elle exécute **sudo** à partir du compte d'utilisateur qui a exécuté la commande Ansible, pas **remote\_user**. Cela peut entraîner une confusion si les deux utilisateurs ont des privilèges **sudo** différents.

Pour vous assurer que vous vous connectez à **localhost** en utilisant SSH comme les autres hôtes gérés, vous pouvez le lister dans votre inventaire. Mais cela l'inclura dans les groupes **all** et **ungrouped**, ce que vous souhaiterez peut-être éviter.

Vous pouvez également modifier le protocole utilisé pour se connecter à **localhost**. La meilleure façon de procéder est de définir la **ansible\_connection variable d'hôte** pour **localhost**. Pour ce faire, dans le répertoire à partir duquel vous exécutez les commandes Ansible, créez un sous-répertoire **host\_vars**. Dans ce sous-répertoire, créez un fichier nommé **localhost** qui contient la ligne **ansible\_connection: smart**. Cela permet de s'assurer que le protocole de connexion **smart** (SSH) est utilisé à la place de **local** pour **localhost**.

Vous pouvez également utiliser cela en sens inverse. Si `127.0.0.1` est listé dans votre inventaire, vous vous connecterez par défaut via `smart`. Vous pouvez également créer un fichier `host_vars/127.0.0.1` contenant la ligne `ansible_connection: local` pour qu'il utilise plutôt `local`.

Les variables d'hôtes sont traitées plus en détail ultérieurement dans le cours.



### NOTE

Vous pouvez également utiliser les *variables de groupe* pour changer le type de connexion pour un groupe d'hôtes complet. Pour cela, placez des fichiers portant le même nom que le groupe dans un répertoire `group_vars` et vérifiez que ces fichiers contiennent des paramètres pour les variables de connexion.

Par exemple, vous souhaiterez peut-être que tous vos hôtes gérés Microsoft Windows utilisent le protocole `winrm` et le port `5986` pour les connexions. Pour configurer ceci, vous pouvez placer tous ces hôtes gérés dans le groupe `windows`, puis créer un fichier nommé `group_vars/windows` contenant les lignes suivantes :

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

## COMMENTAIRES SUR LE FICHIER DE CONFIGURATION

Il y a deux caractères de commentaire autorisés par les fichiers de configuration Ansible : le signe de hachage ou numérique (#) et le point-virgule (;).

Le caractère numérique au début d'une ligne met toute la ligne en commentaire. Il ne doit pas être sur la même ligne qu'une directive.

Le caractère du point-virgule met en commentaire tout ce qui se trouve à sa droite sur la ligne. Il peut se trouver sur la même ligne qu'une directive, tant que celle-ci reste placée à sa gauche.



### RÉFÉRENCES

Pages du manuel `ansible(1)`, `ansible-config(1)`, `ssh-keygen(1)` et `ssh-copy-id(1)`

### Fichier de configuration : Documentation Ansible

[https://docs.ansible.com/ansible/2.7/installation\\_guide/intro\\_configuration.html](https://docs.ansible.com/ansible/2.7/installation_guide/intro_configuration.html)

## ► EXERCICE GUIDÉ

# GESTION DES FICHIERS DE CONFIGURATION ANSIBLE

Dans le cadre de cet exercice, vous allez personnaliser votre environnement Ansible en modifiant un fichier de configuration Ansible.

## RÉSULTATS

Vous devez pouvoir créer un fichier de configuration pour définir votre environnement Ansible avec des paramètres personnalisés persistants.

Connectez-vous en tant qu'utilisateur student sur workstation, puis exécutez **lab deploy-manage setup**. Ce script s'assure que l'hôte géré, servera, est accessible sur le réseau.

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

- 1. Créez le répertoire **/home/student/deploy-manage**, qui contiendra les fichiers pour cet exercice. Accédez à ce nouveau répertoire.

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

- 2. Dans votre répertoire **/home/student/deploy-manage**, utilisez un éditeur de texte pour commencer à modifier un nouveau fichier **ansible.cfg**.

Créez une section **[defaults]** dans ce fichier. Dans cette section, ajoutez une ligne qui utilise la directive **inventory** pour spécifier le fichier **./inventory** comme inventaire par défaut.

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

Enregistrez vos modifications, puis quittez l'éditeur de texte.

- 3. Dans le répertoire **/home/student/deploy-manage**, utilisez un éditeur de texte pour commencer à modifier le nouveau fichier d'inventaire statique, **inventory**.

L'inventaire statique doit contenir trois groupes d'hôtes :

- **[myself]** doit contenir l'hôte localhost.
- **[intranetweb]** doit contenir l'hôte servera.lab.example.com.
- **[everyone]** doit contenir les groupes d'hôtes **myself** et **intranetweb**.

- 3.1. Dans **/home/student/deploy-manage/inventory**, créez le groupe d'hôtes **myself** en ajoutant les lignes ci-après :

```
[myself]
localhost
```

- 3.2. Dans **/home/student/deploy-manage/inventory**, créez le groupe d'hôtes **intranetweb** en ajoutant les lignes ci-après :

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

- 3.3. Dans **/home/student/deploy-manage/inventory**, créez le groupe d'hôtes **everyone** en ajoutant les lignes ci-après :

```
[everyone:children]
myself
intranetweb
```



#### NOTE

Rappelez-vous que vous n'avez pas besoin de créer un groupe spécial pour pouvoir sélectionner tous les hôtes du fichier d'inventaire ; vous pouvez utiliser le groupe d'hôtes **all**. Nous faisons cela pour pratiquer la création de groupes de groupes.

- 3.4. Confirmez que votre fichier **inventory** final ressemble à ce qui suit :

```
[myself]
localhost

[intranetweb]
servera.lab.example.com

[everyone:children]
myself
intranetweb
```

Enregistrez vos modifications, puis quittez l'éditeur de texte.

- 4. Utilisez la commande **ansible** avec l'option **--list-hosts** pour tester la configuration des groupes d'hôtes de votre fichier d'inventaire. Cela ne va pas réellement permettre de se connecter à ces hôtes.

```
[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 everyone --list-hosts
  hosts (2):
    localhost
    servera.lab.example.com
```

- 5. Ouvrez le fichier **/home/student/deploy-manage/ansible.cfg** dans un éditeur de texte. Ajoutez une section **[privilegeEscalation]** de sorte qu'Ansible soit configuré pour utiliser automatiquement la commande **sudo** qui va permettre le basculement de **student** vers **root** lors de l'exécution de tâches sur les hôtes gérés. Vous devez également configurer Ansible afin qu'il vous invite à saisir le mot de passe utilisé par **student** pour exécuter la commande **sudo**.

- 5.1. Créez la section **[privilegeEscalation]** dans le fichier de configuration **/home/student/deploy-manage/ansible.cfg** en ajoutant l'entrée suivante :

```
[privilegeEscalation]
```

- 5.2. Activez l'augmentation des privilèges en définissant la directive **become** sur **true**.

```
become = true
```

- 5.3. Définissez l'augmentation des privilèges pour utiliser la commande **sudo** en définissant la directive **becomeMethod** sur **sudo**.

```
becomeMethod = sudo
```

- 5.4. Définissez l'augmentation des privilèges en définissant la directive **becomeUser** sur **root**.

```
becomeUser = root
```

- 5.5. Activez l'invitation à saisir le mot de passe d'augmentation des privilèges en définissant la directive **becomeAskPass** sur **true**.

```
becomeAskPass = true
```

- 5.6. Confirmez que le fichier **ansible.cfg** final ressemble à ce qui suit :

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

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

Enregistrez vos modifications, puis quittez l'éditeur de texte.

- ▶ 6. Exécutez à nouveau la commande **ansible --list-hosts** afin de vérifier que vous êtes désormais invité à saisir le mot de passe **sudo**.

Lorsque vous êtes invité à entrer le mot de passe sudo, entrez **student**, même s'il n'est pas utilisé pour cet essai à blanc.

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

## Nettoyage

À partir de **workstation**, exécutez le script **lab deploy-manage cleanup** pour effacer cet exercice.

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

L'exercice guidé est maintenant terminé.

# EXÉCUTION DE COMMANDES AD HOC

## OBJECTIFS

Au terme de cette section, les stagiaires doivent pouvoir exécuter une seule tâche d'automatisation Ansible à l'aide d'une commande ad hoc et expliquer certains cas d'utilisation de commandes ad hoc.

Figure 2.0: Exécution de commandes ad hoc

## EXÉCUTION DE COMMANDES AD HOC AVEC ANSIBLE

Une *commande ad hoc* est un moyen d'exécuter rapidement une tâche Ansible unique, que vous n'avez pas besoin de sauvegarder pour l'exécuter plus tard. Ce sont des opérations simples, en ligne, qui peuvent être exécutées sans écrire de playbook.

Les commandes ad hoc sont utiles pour les tests et les modifications rapides. Par exemple, vous pouvez utiliser une commande ad hoc pour vous assurer qu'une certaine ligne existe dans le fichier **/etc/hosts** sur un groupe de serveurs. Vous pouvez utiliser une autre commande ad hoc pour redémarrer efficacement un service sur de nombreuses machines différentes, ou pour vous assurer qu'un paquetage logiciel particulier est à jour.

Les commandes ad hoc sont très utiles pour effectuer rapidement des tâches simples avec Ansible. Elles ont leurs limites, et en général, vous aurez besoin d'utiliser des playbooks Ansible pour vous rendre compte de la pleine puissance d'Ansible. Dans de nombreuses situations, cependant, les commandes ad hoc sont exactement l'outil dont vous aurez besoin pour effectuer des tâches simples rapidement.

### Exécution de commandes Ad Hoc

Utilisez la commande **ansible** pour exécuter des commandes ad hoc :

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

L'argument *host-pattern* est utilisé pour spécifier les hôtes gérés sur lesquels la commande ad hoc doit être exécutée. Il peut s'agir d'un hôte géré spécifique ou d'un groupe d'hôtes dans l'inventaire. Vous avez déjà vu cela utilisé conjointement avec l'option **--list-hosts**, qui vous montre quels hôtes correspondent à un modèle d'hôte particulier. Vous avez également déjà vu que vous pouvez utiliser l'option **-i** pour spécifier un autre emplacement d'inventaire à utiliser que celui par défaut dans le fichier de configuration Ansible actuel.

L'option **-m** prend comme argument le nom du *module* qu'Ansible doit exécuter sur les hôtes ciblés. Les modules sont de petits programmes qui sont exécutés pour implémenter votre tâche. Certains modules n'ont pas besoin d'informations supplémentaires, mais d'autres exigent des arguments supplémentaires pour spécifier les détails de leur fonctionnement. L'option **-a** prend une liste de ces arguments sous la forme d'une chaîne entre guillemets.

L'une des commandes ad hoc les plus simples utilise le module **ping**. Ce module n'effectue pas de ping ICMP, mais vérifie si vous pouvez exécuter des modules basés sur Python sur des hôtes gérés.

Par exemple, la commande ad hoc suivante détermine si tous les hôtes gérés dans l'inventaire peuvent exécuter des modules standard :

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

## Exécution de tâches avec des modules utilisant des commandes ad hoc

Les modules sont les outils que les commandes ad hoc utilisent pour accomplir des tâches. Ansible fournit des centaines de modules qui font des choses différentes. Vous pouvez généralement trouver un module spécial testé qui fait ce dont vous avez besoin dans le cadre de l'installation standard.

La commande **ansible-doc -l** liste tous les modules installés sur le système. Vous pouvez ensuite utiliser **ansible-doc** pour afficher la documentation de modules particuliers par nom, et trouver des informations sur les arguments que les modules prennent comme options. Par exemple, la commande suivante affiche la documentation du module **ping** :

```
[user@controlnode ~]$ ansible-doc ping
> PING      (/usr/lib/python2.7/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 log in 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.

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]

NOTES:
* For Windows targets, use the [win_ping] module instead.
* For Network targets, use the [net_ping] module instead.

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: string
  sample: pong

```

Pour en savoir plus sur les modules, accédez à la documentation en ligne d'Ansible sur [http://docs.ansible.com/ansible/2.7/modules/modules\\_by\\_category.html](http://docs.ansible.com/ansible/2.7/modules/modules_by_category.html).

Le tableau suivant liste un certain nombre de modules utiles à titre d'exemples. Beaucoup d'autres existent.

## Modules Ansible

| CATÉGORIE DE MODULE  | MODULES   |
|----------------------|---|
| Modules de fichiers  | <ul style="list-style-type: none"> <li>copy : copie un fichier local sur l'hôte géré</li> <li>file : définit les autorisations et autres propriétés des fichiers</li> <li>lineinfile : s'assure qu'une ligne particulière figure ou non dans un fichier</li> <li>synchronize : synchronise le contenu en utilisant <b>rsync</b></li> </ul>  |
| Modules de logiciels | <ul style="list-style-type: none"> <li>paquetage : gérer les paquetages à l'aide d'un gestionnaire de paquetages autodétecté, natif du système d'exploitation</li> <li>yum : gérer les paquetages à l'aide du gestionnaire de paquetages YUM</li> <li>apt : gérer les paquetages à l'aide du gestionnaire de paquetages APT</li> <li>dnf : gérer les paquetages à l'aide du gestionnaire de paquetages DNF</li> <li>gem : gérer les gemmes Ruby</li> <li>pip : gérer les paquetages Python à partir de PyPI</li> <li>yum : gérer les paquetages à l'aide du gestionnaire de paquetages YUM</li> </ul> |
| Modules système      | <ul style="list-style-type: none"> <li>firewalld : gérer les ports/services arbitraires en utilisant <b>firewalld</b></li> <li>reboot : redémarrer une machine</li> <li>service : gérer les services</li> <li>user : ajouter, supprimer et gérer des comptes d'utilisateurs</li> </ul>  |
| Modules Net Tools    | <ul style="list-style-type: none"> <li>get_url : télécharger des fichiers via HTTP, HTTPS ou FTP</li> <li>nmcli : gérer le réseautage</li> <li>uri : interagir avec les services Web</li> </ul>   |

La plupart des modules font appel à des arguments. Vous trouverez la liste des arguments disponibles pour un module dans la documentation du module. Les commandes ad hoc transmettent des arguments aux modules à l'aide de l'option **-a**. Lorsqu'aucun argument n'est nécessaire, n'indiquez pas l'option **-a** dans la commande ad hoc. Pour spécifier plusieurs arguments, fournissez-les dans une liste de noms séparés par des espaces.

Par exemple, la commande ad hoc suivante utilise le module user pour s'assurer que l'utilisateur newbie existe et qu'il dispose de l'UID 4000 sur `servera.lab.example.com` :

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

La plupart des modules sont *idempotents*, ce qui signifie qu'ils peuvent être exécutés en toute sécurité plusieurs fois, et que si le système est déjà à l'état correct, ils ne feront rien. Par exemple, si vous réexécuter la commande ad hoc précédente et elle ne devrait signaler aucun changement :

```
[user@controlnode ~]$ ansible -m user -a 'name=newbie uid=4000 state=present' \
> servera.lab.example.com
servera.lab.example.com | SUCCESS => {
    "append": false,
    "changed": false
    "comment": "",
    "group": 4000,
    "home": "/home/newbie",
    "move_home": false,
    "name": "newbie",
    "shell": "/bin/bash",
    "state": "present",
    "uid": 4000
}
```

## Exécution de commandes arbitraires sur des hôtes gérés

Le module command permet aux administrateurs d'exécuter des commandes arbitraires sur la ligne de commande des hôtes gérés. La commande à exécuter est spécifiée en tant qu'argument pour le module en utilisant l'option **-a**. Par exemple, la commande suivante exécute **hostname** sur les hôtes gérés référencés par le modèle hôte `mymanagedhosts`.

```
[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
```

L'exemple de commande ad hoc précédent a renvoyé deux lignes de résultat pour chaque hôte géré. La première ligne est un rapport d'état, qui indique le nom de l'hôte géré sur lequel

l'opération ad hoc a été exécutée, ainsi que le résultat de l'opération. La seconde ligne est le résultat de la commande exécutée à distance à l'aide du module `command` d'Ansible.

Pour une plus grande lisibilité et une meilleure analyse du résultat de la commande ad hoc, les administrateurs peuvent trouver utile d'avoir une seule ligne de résultat pour chaque opération effectuée sur un hôte géré. Utilisez l'option `-o` pour afficher le résultat des commandes ad hoc d'Ansible dans un format de ligne simple.

```
[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
```

Le module `command` permet aux administrateurs d'exécuter rapidement des commandes distantes sur les hôtes gérés. Ces commandes ne sont pas traitées par le shell sur les hôtes gérés. Par conséquent, elles ne peuvent pas accéder aux variables d'environnement shell ni effectuer des opérations shell comme la redirection et le piping.

Pour les situations dans lesquelles les commandes ont besoin d'un traitement shell, les administrateurs peuvent utiliser le module `shell`. Comme pour le module `command`, vous passez les commandes à exécuter sous forme d'arguments à un module dans la commande ad hoc. Ansible exécute ensuite la commande à distance sur les hôtes gérés. Contrairement au module `command`, les commandes sont traitées à travers un shell sur les hôtes gérés. Par conséquent, les variables de l'environnement shell sont accessibles et les opérations telles que la redirection et le piping seront également disponibles.

L'exemple suivant illustre la différence entre les modules `command` et `shell`. Si vous essayez d'exécuter la commande Bash intégrée `set` avec ces deux modules, seul le module `shell` aboutit.

```
[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:interact
ive_comments:progcomp:promptvars:sourcepath
BASH_ALIASES=()
...output omitted...
```

À la fois les modules `command` et `shell` requièrent une installation Python fonctionnelle sur l'hôte géré. Un troisième module, `raw`, peut exécuter des commandes directement à l'aide du shell distant, en contournant le sous-système du module. Ceci est utile lors de la gestion des systèmes sur lesquels Python ne peut pas être installé (par exemple, un routeur réseau). Il peut également être utilisé pour installer Python sur un hôte.

**IMPORTANT**

Dans la plupart des cas, il est recommandé d'éviter les modules « run command » `command`, `shell` et `raw`.

La plupart des autres modules sont idempotents et peuvent effectuer le suivi des modifications automatiquement. Ils peuvent tester l'état des systèmes et ne rien faire si ces derniers présentent déjà l'état correct. En revanche, il est beaucoup plus compliqué d'utiliser les modules « run command » d'une manière idempotente. Si vous dépendez d'eux, il sera peut-être plus difficile pour vous d'être certain que la réexécution d'une commande ad hoc ou d'un playbook ne provoquera pas un échec inattendu. Lorsqu'un module `shell` ou `command` s'exécute, il rapporte généralement un état **CHANGED** selon que, selon lui, cela affecte ou non l'état de la machine.

Dans certains cas, les modules « run command » sont des outils précieux et une solution appropriée à un problème. Si vous n'avez pas besoin de les utiliser, il est probablement préférable d'essayer d'abord d'utiliser le module `command`, en ayant recours aux modules `shell` ou `raw` uniquement si vous avez besoin de leurs fonctions spéciales.

## CONFIGURATION DE CONNEXIONS POUR LES COMMANDES AD HOC

Les directives pour les connexions aux hôtes gérés et l'augmentation des privilèges peuvent être configurés dans le fichier de configuration Ansible, et elles peuvent également l'être à l'aide d'options dans des commandes ad hoc. Lorsqu'elles sont définies à l'aide d'options dans les commandes ad hoc, elles prévalent sur la directive configurée dans le fichier de configuration d'Ansible. Le tableau suivant montre les options de ligne de commande analogique pour chaque directive du fichier de configuration.

### Options de ligne de commande Ansible

| DIRECTIVES DU FICHIER DE CONFIGURATION | OPTION DE LIGNE DE COMMANDE        |
|--|------------------------------------|
| <code>inventory</code>                 | <code>-i</code>                    |
| <code>remote_user</code>               | <code>-u</code>                    |
| <code>become</code>                    | <code>--become, -b</code>          |
| <code>become_method</code>             | <code>--become-method</code>       |
| <code>become_user</code>               | <code>--become-user</code>         |
| <code>become_ask_pass</code>           | <code>--ask-become-pass, -K</code> |

Avant de configurer ces directives à l'aide d'options de ligne de commande, leurs valeurs en cours peuvent être déterminées en consultant le résultat de la commande `ansible --help`.

```
[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)
```



## RÉFÉRENCES

Page de manuel (1)[ansible](#)

**Utilisation de modèles : Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/user\\_guide/intro\\_patterns.html](https://docs.ansible.com/ansible/2.7/user_guide/intro_patterns.html)

**Introduction aux commandes ad hoc : Documentation Ansible**

[http://docs.ansible.com/ansible/2.7/user\\_guide/intro\\_adhoc.html](http://docs.ansible.com/ansible/2.7/user_guide/intro_adhoc.html)

**Index des modules : Documentation Ansible**

[http://docs.ansible.com/ansible/2.7/modules/modules\\_by\\_category](http://docs.ansible.com/ansible/2.7/modules/modules_by_category)

**command : exécute une commande sur un nœud distant : Documentation Ansible**

[http://docs.ansible.com/ansible/2.7/modules/command\\_module.html](http://docs.ansible.com/ansible/2.7/modules/command_module.html)

**shell : exécute des commandes dans des nœuds : Documentation Ansible**

[http://docs.ansible.com/ansible/2.7/modules/shell\\_module.html](http://docs.ansible.com/ansible/2.7/modules/shell_module.html)

## ► EXERCICE GUIDÉ

# EXÉCUTION DE COMMANDES AD HOC

Dans cet exercice, vous allez exécuter des commandes ad hoc sur plusieurs hôtes gérés.

## RÉSULTATS

Vous devez pouvoir exécuter des commandes sur des hôtes gérés sur une base ad hoc grâce à l'augmentation des privilèges.

Vous allez exécuter des commandes ad hoc sur `workstation` et `servera` en utilisant le compte utilisateur `devops`. Ce compte a la même configuration `sudo` sur `workstation` et `servera`.

Connectez-vous en tant qu'utilisateur `student` sur `workstation`, puis exécutez `lab deploy-adhoc setup`. Ce script s'assure que l'hôte géré, `servera`, est accessible sur le réseau. Il va également créer et remplir le répertoire de travail `/home/student/deploy-adhoc` avec les éléments utilisés dans cet exercice.

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

- ▶ 1. Déterminez la configuration `sudo` pour le compte `devops` à la fois sur `workstation` et `servera`.
  - 1.1. Déterminez la configuration `sudo` pour le compte `devops` qui a été configuré au moment de la conception de `workstation`. Saisissez `student` si vous êtes invité à entrer le mot de passe du compte `student`.

```
[student@workstation ~]$ sudo cat /etc/sudoers.d/devops
[sudo] password for student: student
devops ALL=(ALL) NOPASSWD: ALL
```

Notez que l'utilisateur dispose de tous les privilèges `sudo` mais qu'il ne requiert aucune authentification par mot de passe.

- 1.2. Déterminez la configuration `sudo` pour le compte `devops` qui a été configuré au moment de la conception de `servera`.

```
[student@workstation ~]$ ssh devops@servera.lab.example.com
[devops@servera ~]$ sudo cat /etc/sudoers.d/devops
devops ALL=(ALL) NOPASSWD: ALL
[devops@servera ~]$ exit
```

Notez que l'utilisateur dispose de tous les privilèges `sudo` mais qu'il ne requiert aucune authentification par mot de passe.

- ▶ 2. Remplacez le répertoire par **/home/student/deploy-adhoc** et examinez le contenu des fichiers **ansible.cfg** et **inventory**.

```
[student@workstation ~]$ cd /home/student/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
```

Le fichier de configuration utilise le fichier **inventory** du répertoire en tant qu'inventaire Ansible. Notez qu'Ansible n'est pas encore configuré pour utiliser l'augmentation des priviléges.

- ▶ 3. À l'aide du groupe d'hôtes **all** et du module **ping**, exécutez une commande ad hoc qui permet de s'assurer que tous les hôtes gérés puissent exécuter des modules Ansible à l'aide de Python.

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

- ▶ 4. À l'aide du module **command**, exécutez une commande ad hoc sur **workstation** afin d'identifier le compte utilisateur utilisé par Ansible pour effectuer des opérations sur des hôtes gérés. Utilisez le modèle hôte **localhost** afin de vous connecter à **workstation** pour l'exécution de la commande ad hoc. Comme vous êtes connecté en local, **workstation** est à la fois le nœud de contrôle et l'hôte géré.

```
[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
```

Notez que la commande ad hoc a été effectuée sur l'hôte géré en tant qu'utilisateur **student**.

- ▶ 5. Exécutez la commande ad hoc précédente sur **workstation** mais connectez-vous et réalisez l'opération avec le compte utilisateur **devops** en utilisant l'option **-u**.

```
[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
```

Notez que la commande ad hoc a été effectuée sur l'hôte géré en tant qu'utilisateur devops.

- ▶ 6. À l'aide du module `command`, exécutez une commande ad hoc sur `workstation` pour afficher le contenu du fichier `/etc/motd`. Exécutez la commande en utilisant le compte devops.

```
[student@workstation deploy-adhoc]$ ansible localhost -m command \
> -a 'cat /etc/motd' -u devops
localhost | CHANGED | rc=0 >>
```

Notez que le fichier `/etc/motd` est actuellement vide.

- ▶ 7. À l'aide du module `copy`, exécutez une commande ad hoc sur `workstation` pour modifier le contenu du fichier `/etc/motd` de sorte qu'il se compose de la chaîne « Managed by Ansible » suivie d'une nouvelle ligne. Exécutez la commande en utilisant le compte devops, mais n'utilisez pas l'option `--become` pour basculer vers root. La commande ad hoc doit échouer en raison du manque d'autorisations.

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

La commande ad hoc a échoué car l'utilisateur devops ne dispose pas de droits en écriture sur le fichier.

- ▶ 8. Exécutez à nouveau la commande en utilisant une élévation de priviléges. Vous pouvez corriger les paramètres dans le fichier `ansible.cfg`, mais pour cet exemple, utilisez simplement les options de ligne de commande appropriées de la commande `ansible`. À l'aide du module `copy`, exécutez la commande précédente sur `workstation` pour modifier le contenu du fichier `/etc/motd` de sorte qu'il se compose de la chaîne « Managed by Ansible » suivie d'une nouvelle ligne. Via l'utilisateur devops, établissez la connexion sur l'hôte géré, mais effectuez l'opération en tant qu'utilisateur root avec l'option `--become`. L'utilisation de l'option `--become` est suffisante car la valeur par défaut pour la directive `become_user` est définie sur root dans le fichier `/etc/ansible/ansible.cfg`.

```
[student@workstation deploy-adhoc]$ ansible localhost -m copy \
> -a 'content="Managed by Ansible\n" dest=/etc/motd' -u devops --become
localhost | CHANGED => {
  "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-1463518320.68-167292050637471/
source",
"state": "file",
"uid": 0
}
```

Notez que la commande a réussi cette fois car la commande ad hoc a été exécutée avec l'augmentation des priviléges.

- 9. Exécutez à nouveau la commande ad hoc précédente sur tous les hôtes à l'aide du groupe d'hôtes `all`. Cela permet de s'assurer que `/etc/motd` à la fois sur `workstation` et `servera` se compose du texte « `Managed by Ansible` ».

```
[student@workstation deploy-adhoc]$ ansible all -m copy \
> -a 'content="Managed by Ansible\n" dest=/etc/motd' -u devops --become
localhost | SUCCESS => {
    "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
}
servera.lab.example.com | CHANGED => {
    "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-1541518645.68-122144769062037/
source",
    "state": "file",
    "uid": 0
}
```

Vous devriez voir **SUCCESS** pour `localhost` et **CHANGED** pour `servera`. Toutefois, `localhost` doit indiquer `"changed": false` car le fichier présente déjà l'état correct. À l'inverse, `servera` doit indiquer `"changed": true` car la commande ad hoc a mis à jour le fichier à l'état correct.

- 10. À l'aide du module `command`, exécutez une commande ad hoc pour exécuter `cat /etc/motd` afin de vérifier que le contenu du fichier a bien été modifié à la fois sur `workstation` et `servera`. Utilisez le groupe d'hôtes `all` et l'utilisateur `devops` pour spécifier et effectuer la connexion aux hôtes gérés. Vous n'avez pas besoin d'augmenter les privilèges pour que cette commande fonctionne.

```
[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
```

## Nettoyage

À partir de `workstation`, exécutez le script `lab deploy-adhoc cleanup` pour effacer cet exercice.

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

L'exercice guidé est maintenant terminé.

## ► OPEN LAB

# DÉPLOIEMENT D'ANSIBLE

## LISTE DE CONTRÔLE DES PERFORMANCES

Dans cet atelier, vous allez configurer un nœud de contrôle Ansible pour les connexions aux hôtes d'inventaire et utiliser des commandes ad hoc pour réaliser des actions sur des hôtes gérés.

## RÉSULTATS

Vous devez pouvoir configurer un nœud de contrôle pour exécuter des commandes ad hoc sur des hôtes gérés.

Vous allez utiliser Ansible pour gérer un certain nombre d'hôtes depuis `workstation.lab.example.com` en tant qu'utilisateur `student`. Vous allez configurer un répertoire de projet contenant un fichier `ansible.cfg` avec des valeurs par défaut spécifiques, et un répertoire `inventory` contenant un fichier d'inventaire.

Vous allez utiliser des commandes ad hoc pour vous assurer que le fichier `/etc/motd` sur tous les hôtes gérés est constitué du contenu spécifié.

Connectez-vous en tant qu'utilisateur `student` sur `workstation`, puis exécutez `lab deploy-review setup`. Ce script permet de s'assurer que les hôtes gérés sont accessibles sur le réseau.

```
[student@workstation ~]$ lab deploy-review setup
```

1. Vérifiez que le paquetage `ansible` est installé sur le nœud de contrôle et exécutez la commande `ansible --version`.
2. Dans le répertoire personnel de l'utilisateur `student` sur `workstation`, `/home/student`, créez un nouveau répertoire nommé `deploy-review`. Choisissez ce répertoire.
3. Créez un fichier `ansible.cfg` dans le répertoire `deploy-review`, que vous allez utiliser pour définir les valeurs par défaut Ansible suivantes :
  - Connectez-vous aux hôtes gérés en tant qu'utilisateur `devops`.
  - Utilisez le sous-répertoire `inventory` pour contenir le fichier d'inventaire.
  - Désactivez l'augmentation des privilèges par défaut. Si l'escalade des privilèges est activée à partir de la ligne de commande, configurez les paramètres par défaut pour qu'Ansible utilise la méthode `sudo` afin de basculer vers le compte utilisateur `root`. Ansible ne doit pas demander le mot de passe de connexion `devops` ni le mot de passe `sudo`.

Les hôtes gérés ont déjà été configurés avec un utilisateur `devops` qui peut se connecter avec l'authentification par clé SSH, et qui peut exécuter n'importe quelle commande en tant qu'utilisateur `root` en utilisant la commande `sudo` sans mot de passe.

4. Créez le répertoire `/home/student/deploy-review/inventory`.

Téléchargez le fichier `http://materials.example.com/labs/deploy-review/inventory` et enregistrez-le en tant que fichier d'inventaire statique nommé `/home/student/deploy-review/inventory/inventory`.

5. Exécutez une commande ad hoc ciblant le groupe d'hôtes `a11` pour vérifier que `devops` est l'utilisateur distant et que l'augmentation des priviléges est désactivée par défaut.
6. Exécutez une commande ad hoc, ciblant le groupe d'hôtes `a11`, qui utilise le module `copy` pour modifier le contenu du fichier `/etc/motd` sur tous les hôtes.  
Utilisez l'option `content` du module `copy` pour vous assurer que le fichier `/etc/motd` comporte la chaîne « `This server is managed by Ansible.\n` » sous la forme d'une ligne unique. (Le `\n` utilisé avec la directive `content` oblige le module à insérer une nouvelle ligne à la fin de la chaîne.)  
Vous devez demander l'augmentation de priviléges à partir de la ligne de commande pour que cela fonctionne avec vos valeurs par défaut `ansible.cfg` actuelles.
7. Si vous réexécutez la même commande ad hoc, vous devriez voir que le module `copy` détecte que les fichiers sont déjà corrects et qu'il ne les modifie pas. Recherchez la commande ad hoc pour signaler `SUCCESS` et la ligne "`changed": "false`" pour chaque hôte géré.
8. Pour confirmer cela autrement, exécutez une commande ad hoc qui cible le groupe d'hôtes `a11` et qui utilise le module `command` pour exécuter la commande `cat /etc/motd`. Le résultat de la commande `ansible` doit afficher la chaîne « `This server is managed by Ansible.` » pour tous les hôtes. Vous n'avez pas besoin d'augmenter les priviléges pour cette commande ad hoc.
9. Exécutez `lab deploy-review grade` sur `workstation` pour vérifier votre travail.

```
[student@workstation deploy-review]$ lab deploy-review grade
```

## Nettoyage

Sur `workstation`, exécutez le script `lab deploy-review cleanup` pour effacer les ressources créées au cours de cet atelier.

```
[student@workstation ~]$ lab deploy-review cleanup
```

L'atelier est maintenant terminé.

## ► SOLUTION

# DÉPLOIEMENT D'ANSIBLE

## LISTE DE CONTRÔLE DES PERFORMANCES

Dans cet atelier, vous allez configurer un nœud de contrôle Ansible pour les connexions aux hôtes d'inventaire et utiliser des commandes ad hoc pour réaliser des actions sur des hôtes gérés.

## RÉSULTATS

Vous devez pouvoir configurer un nœud de contrôle pour exécuter des commandes ad hoc sur des hôtes gérés.

Vous allez utiliser Ansible pour gérer un certain nombre d'hôtes depuis `workstation.lab.example.com` en tant qu'utilisateur `student`. Vous allez configurer un répertoire de projet contenant un fichier `ansible.cfg` avec des valeurs par défaut spécifiques, et un répertoire `inventory` contenant un fichier d'inventaire.

Vous allez utiliser des commandes ad hoc pour vous assurer que le fichier `/etc/motd` sur tous les hôtes gérés est constitué du contenu spécifié.

Connectez-vous en tant qu'utilisateur `student` sur `workstation`, puis exécutez `lab deploy-review setup`. Ce script permet de s'assurer que les hôtes gérés sont accessibles sur le réseau.

```
[student@workstation ~]$ lab deploy-review setup
```

- Vérifiez que le paquetage `ansible` est installé sur le nœud de contrôle et exécutez la commande `ansible --version`.

- Vérifiez que le paquetage `ansible` est installé.

```
[student@workstation ~]$ yum list installed ansible
Loaded plugins: langpacks, search-disabled-repos
Installed Packages
ansible.noarch      2.7.1-1.el7ae      @ansible
```

- Exécutez la commande `ansible --version` pour confirmer la version d'Ansible installée.

```
[student@workstation ~]$ ansible --version
ansible 2.7.1
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/home/student/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/site-packages/ansible
  executable location = /usr/bin/ansible
  python version = 2.7.5 (default, Sep 12 2018, 05:31:16) [GCC 4.8.5 20150623 (Red Hat 4.8.5-36)]
```

2. Dans le répertoire personnel de l'utilisateur student sur workstation, **/home/student**, créez un nouveau répertoire nommé **deploy-review**. Choisissez ce répertoire.

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

3. Créez un fichier **ansible.cfg** dans le répertoire **deploy-review**, que vous allez utiliser pour définir les valeurs par défaut Ansible suivantes :
- Connectez-vous aux hôtes gérés en tant qu'utilisateur devops.
  - Utilisez le sous-répertoire **inventory** pour contenir le fichier d'inventaire.
  - Désactivez l'augmentation des privilèges par défaut. Si l'escalade des privilèges est activée à partir de la ligne de commande, configurez les paramètres par défaut pour qu'Ansible utilise la méthode **sudo** afin de basculer vers le compte utilisateur root. Ansible ne doit pas demander le mot de passe de connexion devops ni le mot de passe **sudo**.

Les hôtes gérés ont déjà été configurés avec un utilisateur devops qui peut se connecter avec l'authentification par clé SSH, et qui peut exécuter n'importe quelle commande en tant qu'utilisateur root en utilisant la commande **sudo** sans mot de passe.

- 3.1. Utilisez un éditeur de texte pour créer le fichier **/home/student/deploy-review/ansible.cfg**. Créez une section **[defaults]**. Ajoutez une directive **remote\_user** pour qu'Ansible fasse appel à l'utilisateur devops lors de la connexion aux hôtes gérés. Ajoutez une directive **inventory** pour configurer Ansible afin d'utiliser le répertoire **/home/student/deploy-review/inventory** comme emplacement par défaut pour le fichier d'inventaire.

```
[defaults]
remote_user = devops
inventory = inventory
```

- 3.2. Dans le fichier **/home/student/deploy-review/ansible.cfg**, créez la section **[privilege\_escalation]** et ajoutez les entrées suivantes pour désactiver l'augmentation des privilèges. Définissez la méthode d'augmentation des privilèges pour utiliser le compte **root** avec **sudo** et sans authentification par mot de passe.

```
[privilege_escalation]
become = False
become_method = sudo
become_user = root
become_ask_pass = False
```

- 3.3. Une fois terminé, le fichier **ansible.cfg** doit se présenter comme suit :

```
[defaults]
remote_user = devops
inventory = inventory

[privilege_escalation]
become = False
become_method = sudo
become_user = root
```

```
become_ask_pass = False
```

Enregistrez vos modifications, puis quittez l'éditeur.

**4.** Créez le répertoire **/home/student/deploy-review/inventory**.

Téléchargez le fichier <http://materials.example.com/labs/deploy-review/inventory> et enregistrez-le en tant que fichier d'inventaire statique nommé **/home/student/deploy-review/inventory/inventory**.

4.1. Créez le répertoire **/home/student/deploy-review/inventory**.

```
[student@workstation deploy-review]$ mkdir inventory
```

4.2. Téléchargez le fichier <http://materials.example.com/labs/deploy-review/inventory> dans le répertoire **/home/student/deploy-review/inventory**.

```
[student@workstation deploy-review]$ wget -O inventory/inventory \
> http://materials.example.com/labs/deploy-review/inventory
```

4.3. Inspectez le contenu du fichier **/home/student/deploy-review/inventory/inventory**.

```
[student@workstation deploy-review]$ cat inventory/inventory
[internetweb]
serverb.lab.example.com

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

**5.** Exécutez une commande ad hoc ciblant le groupe d'hôtes **all** pour vérifier que **devops** est l'utilisateur distant et que l'augmentation des privilèges est désactivée par défaut.

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

serverc.lab.example.com | CHANGED | rc=0 >>
uid=1001(devops) gid=1001(devops) groups=1001(devops) context=unconfined_u:
unconfined_r:unconfined_t:s0-s0:c0.c1023

servera.lab.example.com | CHANGED | rc=0 >>
uid=1001(devops) gid=1001(devops) groups=1001(devops) context=unconfined_u:
unconfined_r:unconfined_t:s0-s0:c0.c1023

serverd.lab.example.com | CHANGED | rc=0 >>
uid=1001(devops) gid=1001(devops) groups=1001(devops) context=unconfined_u:
unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Vos résultats peuvent être retournés dans un ordre différent.

6. Exécutez une commande ad hoc, ciblant le groupe d'hôtes `all`, qui utilise le module `copy` pour modifier le contenu du fichier `/etc/motd` sur tous les hôtes.

Utilisez l'option `content` du module `copy` pour vous assurer que le fichier `/etc/motd` comporte la chaîne « `This server is managed by Ansible.\n` » sous la forme d'une ligne unique. (Le `\n` utilisé avec la directive `content` oblige le module à insérer une nouvelle ligne à la fin de la chaîne.)

Vous devez demander l'augmentation de priviléges à partir de la ligne de commande pour que cela fonctionne avec vos valeurs par défaut `ansible.cfg` actuelles.

```
[student@workstation deploy-review]$ ansible all -m copy \
> -a 'content="This server is managed by Ansible.\n" dest=/etc/motd' --become
servera.lab.example.com | CHANGED => {
    "changed": true,
    "checksum": "93d304488245bb2769752b95e0180607effc69ad",
    "dest": "/etc/motd",
    "gid": 0,
    "group": "root",
    "md5sum": "af74293c7b2a783c4f87064374e9417a",
    "mode": "0644",
    "owner": "root",
    "secontext": "system_u:object_r:etc_t:s0",
    "size": 35,
    "src": "/home/devops/.ansible/tmp/ansible-tmp-1499275864.56-280761564717921/
source",
    "state": "file",
    "uid": 0
}
serverb.lab.example.com | CHANGED => {
    "changed": true,
    "checksum": "93d304488245bb2769752b95e0180607effc69ad",
    "dest": "/etc/motd",
    "gid": 0,
    "group": "root",
    "md5sum": "af74293c7b2a783c4f87064374e9417a",
    "mode": "0644",
    "owner": "root",
    "secontext": "system_u:object_r:etc_t:s0",
    "size": 35,
    "src": "/home/devops/.ansible/tmp/ansible-tmp-1499275864.51-224886037138847/
source",
    "state": "file",
    "uid": 0
}
serverc.lab.example.com | CHANGED => {
    "changed": true,
    "checksum": "93d304488245bb2769752b95e0180607effc69ad",
    "dest": "/etc/motd",
    "gid": 0,
    "group": "root",
    "md5sum": "af74293c7b2a783c4f87064374e9417a",
    "mode": "0644",
    "owner": "root",
    "secontext": "system_u:object_r:etc_t:s0",
    "size": 35,
```

```

"src": "/home/devops/.ansible/tmp/ansible-tmp-1499275864.56-242019037094684/
source",
"state": "file",
"uid": 0
}
serverd.lab.example.com | CHANGED => {
    "changed": true,
    "checksum": "93d304488245bb2769752b95e0180607effc69ad",
    "dest": "/etc/motd",
    "gid": 0,
    "group": "root",
    "md5sum": "af74293c7b2a783c4f87064374e9417a",
    "mode": "0644",
    "owner": "root",
    "secontext": "system_u:object_r:etc_t:s0",
    "size": 35,
    "src": "/home/devops/.ansible/tmp/ansible-tmp-1499275864.58-48889952156589/
source",
    "state": "file",
    "uid": 0
}

```

7. Si vous réexécutez la même commande ad hoc, vous devriez voir que le module copy détecte que les fichiers sont déjà corrects et qu'il ne les modifie pas. Recherchez la commande ad hoc pour signaler **SUCCESS** et la ligne "**changed": false**" pour chaque hôte géré.

```

[student@workstation deploy-review]$ ansible all -m copy \
> -a 'content="This server is managed by Ansible.\n" dest=/etc/motd' --become
servera.lab.example.com | SUCCESS => {
    "changed": false,
    "checksum": "93d304488245bb2769752b95e0180607effc69ad",
    "dest": "/etc/motd",
    "gid": 0,
    "group": "root",
    "mode": "0644",
    "owner": "root",
    "path": "/etc/motd",
    "secontext": "system_u:object_r:etc_t:s0",
    "size": 35,
    "state": "file",
    "uid": 0
}
serverd.lab.example.com | SUCCESS => {
    "changed": false,
    "checksum": "93d304488245bb2769752b95e0180607effc69ad",
    "dest": "/etc/motd",
    "gid": 0,
    "group": "root",
    "mode": "0644",
    "owner": "root",
    "path": "/etc/motd",
    "secontext": "system_u:object_r:etc_t:s0",
    "size": 35,
    "state": "file",
    "uid": 0
}

```

```

        "uid": 0
    }
    serverc.lab.example.com | SUCCESS => {
        "changed": false,
        "checksum": "93d304488245bb2769752b95e0180607effc69ad",
        "dest": "/etc/motd",
        "gid": 0,
        "group": "root",
        "mode": "0644",
        "owner": "root",
        "path": "/etc/motd",
        "secontext": "system_u:object_r:etc_t:s0",
        "size": 35,
        "state": "file",
        "uid": 0
    }
    serverb.lab.example.com | SUCCESS => {
        "changed": false,
        "checksum": "93d304488245bb2769752b95e0180607effc69ad",
        "dest": "/etc/motd",
        "gid": 0,
        "group": "root",
        "mode": "0644",
        "owner": "root",
        "path": "/etc/motd",
        "secontext": "system_u:object_r:etc_t:s0",
        "size": 35,
        "state": "file",
        "uid": 0
    }
}

```

- Pour confirmer cela autrement, exéutez une commande ad hoc qui cible le groupe d'hôtes `all` et qui utilise le module `command` pour exécuter la commande `cat /etc/motd`. Le résultat de la commande `ansible` doit afficher la chaîne « **This server is managed by Ansible.** » pour tous les hôtes. Vous n'avez pas besoin d'augmenter les privilèges pour cette commande ad hoc.

```

[student@workstation deploy-review]$ ansible all -m command -a 'cat /etc/motd'
serverb.lab.example.com | CHANGED | rc=0 >>
This server is managed by Ansible.

servera.lab.example.com | CHANGED | rc=0 >>
This server is managed by Ansible.

serverd.lab.example.com | CHANGED | rc=0 >>
This server is managed by Ansible.

serverc.lab.example.com | CHANGED | rc=0 >>
This server is managed by Ansible.

```

- Exédeztez `lab deploy-review grade` sur `workstation` pour vérifier votre travail.

```
[student@workstation deploy-review]$ lab deploy-review grade
```

## Nettoyage

Sur workstation, exéutez le script **lab deploy-review cleanup** pour effacer les ressources créées au cours de cet atelier.

```
[student@workstation ~]$ lab deploy-review cleanup
```

L'atelier est maintenant terminé.

# RÉSUMÉ

---

Dans ce chapitre, vous avez appris les principes suivants :

- Tout système sur lequel Ansible est installé, et ayant accès aux fichiers de configuration et aux playbooks appropriés pour gérer des systèmes distants (*hôtes gérés*) est appelé un *nœud de contrôle*.
- Les hôtes gérés sont définis dans l'*inventaire*. Les modèles hôtes sont utilisés pour faire référence aux hôtes gérés définis dans un inventaire.
- Les inventaires peuvent être des fichiers statiques ou des fichiers générés de manière dynamique par un programme à partir d'une source externe telle qu'un service d'annuaire ou un système de gestion du cloud.
- L'emplacement de l'inventaire est déterminé par le fichier de configuration Ansible utilisé. Cependant, il est généralement conservé avec les fichiers de playbook.
- Ansible recherche son fichier de configuration à plusieurs endroits par ordre de priorité. Le premier fichier de configuration trouvé est utilisé ; tous les autres sont ignorés.
- La commande **ansible** est utilisée pour exécuter des *commandes ad hoc* sur des hôtes gérés.
- Les commandes ad hoc déterminent l'opération à effectuer via l'utilisation de *modules* et de leurs arguments.
- Les commandes ad hoc qui nécessitent des autorisations supplémentaires peuvent utiliser les fonctions d'*augmentation des privilèges* d'Ansible.



## CHAPITRE 3

# MISE EN ŒUVRE DE PLAYBOOKS

### PROJET

Écrire un playbook Ansible simple et l'exécuter pour automatiser les tâches sur plusieurs hôtes.

### OBJECTIFS

- Écrire un playbook Ansible de base et l'exécuter en utilisant la commande **ansible-playbook**.
- Écrire un playbook qui utilise plusieurs plays et l'augmentation des privilèges par play.
- Utiliser efficacement **ansible-doc** pour apprendre à utiliser de nouveaux modules visant à implémenter des tâches pour un play.

### SECTIONS

- Écriture et exécution de playbooks (et exercice guidé)
- Mise en œuvre de plusieurs plays (et exercice guidé)

### ATELIER

- Mise en œuvre de playbooks

# ÉCRITURE ET EXÉCUTION DE PLAYBOOKS

---

## OBJECTIF

À la fin de cette section, les stagiaires doivent pouvoir écrire un playbook Ansible de base et l'exécuter à l'aide de la commande **ansible-playbook**.

Figure 3.0: Mise en œuvre de playbooks Ansible

## PLAYBOOKS ANSIBLE ET COMMANDES AD HOC

Les commandes ad hoc peuvent exécuter une tâche simple unique sur un ensemble d'hôtes ciblés en tant que commande ponctuelle. Toutefois, la puissance réelle d'Ansible réside dans l'utilisation des playbooks pour exécuter plusieurs tâches complexes sur un ensemble d'hôtes ciblés d'une manière facilement reproductible.

Un *play* est un ensemble ordonné de tâches qui doivent être exécutées sur des hôtes choisis de votre inventaire. Un *playbook* est un fichier texte contenant une liste d'un ou plusieurs plays à exécuter dans l'ordre.

Les plays vous permettent de transformer un ensemble complexe et fastidieux de tâches administratives manuelles en opérations de routine facilement reproductibles, dont les résultats sont positifs et prévisibles. Dans un playbook, vous pouvez enregistrer la séquence de tâches d'un play sous une forme lisible par l'homme et immédiatement exécutable. Les tâches elles-mêmes, par la façon dont elles sont écrites, décrivent les étapes nécessaires pour déployer votre application ou infrastructure.

## FORMAT D'UN PLAYBOOK ANSIBLE

Pour vous aider à comprendre le format d'un playbook, nous allons examiner une commande ad hoc que vous avez vue dans un chapitre précédent :

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

Vous pouvez réécrire cette commande comme un play de tâche simple unique et l'enregistrer dans un playbook. Le playbook obtenu ressemble à ce qui suit :

### Exemple 3.1. Un playbook simple

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

Un playbook est un fichier texte écrit au format YAML, dont l'extension est généralement **yml**. Le playbook utilise la mise en retrait avec des espaces pour indiquer la structure de ses données. YAML n'énonce pas d'exigences strictes sur le nombre d'espaces utilisés pour la mise en retrait, mais pose deux règles de base.

- Les éléments de données situés au même niveau hiérarchique (tels que les éléments d'une même liste) doivent adopter la même mise en retrait.
- La mise en retrait des enfants d'un élément doit être plus importante que celle de leurs parents.

Vous pouvez également ajouter des lignes vierges afin d'améliorer la lisibilité.



### IMPORTANT

Seuls les espaces peuvent être utilisés pour la mise en retrait ; les tabulations ne sont pas autorisées.

Si vous utilisez l'éditeur de texte **vi**, vous pouvez appliquer certains paramètres qui peuvent faciliter la modification des playbooks. Par exemple, vous pouvez ajouter la ligne suivante à votre fichier **\$HOME/.vimrc**, et lorsque **vi** détecte que vous modifiez un fichier YAML, il effectue une mise en retrait avec deux espaces lorsque vous appuyez sur la touche **tabulation** et il met automatiquement en retrait les lignes suivantes.

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

Un playbook commence par une ligne composée de trois tirets (---) comme un marqueur de début de document. Il peut se terminer par trois points (...) comme un marqueur de fin de document, ce qui, dans la pratique, est souvent omis.

Entre ces marqueurs, le playbook est défini comme une liste de plays. Un élément d'une liste YAML commence avec un seul tiret suivi d'un espace. Par exemple, une liste YAML peut ressembler à ce qui suit :

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

Dans Exemple 3.1, « Un playbook simple », la ligne après --- commence par un tiret et marque le début du premier (et unique) play de la liste de plays.

Le play en soi est une collection de paires clé-valeur. Les clés du même play doivent présenter la même mise en retrait. L'exemple suivant montre un extrait de code YAML avec trois clés. Les deux premières clés présentent des valeurs simples. La troisième est composée d'une liste de trois éléments en guise de valeur.

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

L'exemple de play original a trois touches, **name**, **hosts** et **tasks**, car ces clés présentent toutes la même mise en retrait.

La première ligne de l'exemple de play commence par un tiret et un espace (indiquant que le play est le premier élément d'une liste). Ensuite, la première clé est l'attribut **name**. La clé **name** associe une chaîne arbitraite au play comme étiquette. Ce dernier sert à identifier l'objet du play. La clé **name** est facultative, toutefois son utilisation est recommandée, car elle permet de donner des indications sur le playbook. Cet attribut se révèle particulièrement utile lorsqu'un playbook contient plusieurs plays.

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

La deuxième clé du play est l'attribut **hosts** qui spécifie les hôtes pour lesquels les tâches du play sont exécutées. Comme pour l'argument de la commande **ansible**, l'attribut **hosts** prend un modèle hôte comme valeur, tel que les noms des hôtes ou groupes gérés dans l'inventaire.

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

Enfin, la dernière clé du play est l'attribut **tasks**, dont la valeur spécifie une liste de tâches à exécuter pour ce play. Cet exemple contient une seule tâche qui exécute le module **user** avec des arguments spécifiques (pour garantir l'existence de l'utilisateur **newbie** et de l'UID 4000).

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

L'attribut **tasks** est la partie du play qui liste dans l'ordre les tâches à exécuter sur les hôtes gérés. Chaque tâche dans la liste est en elle-même une collection de paires clé-valeur.

Dans notre exemple, la seule tâche du play contient deux clés :

- **name** est un libellé facultatif décrivant l'objet de la tâche. Il est judicieux de nommer toutes les tâches pour donner des indications sur l'objet de chaque étape du processus d'automatisation.
- **user** est le module à exécuter pour cette tâche. Ses arguments sont transférés comme une collection de paires clé-valeur qui constituent les enfants du module (**name**, **uid** et **state**).

Voici un autre exemple d'un attribut **tasks** comprenant plusieurs tâches. Il utilise le module **service** pour garantir que plusieurs services réseau sont activés pour être lancés au démarrage :

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

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

**IMPORTANT**

L'ordre dans lequel les plays et les tâches sont listés dans un playbook est important, car Ansible les exécute dans le même ordre.

Les playbooks que nous vous avons présentés jusqu'alors sont des exemples de base. Nous allons vous présenter des exemples d'opérations plus complexes que vous pouvez réaliser avec des plays et des tâches dans la suite de ce cours.

## EXÉCUTION DE PLAYBOOKS

La commande **ansible-playbook** sert à exécuter des playbooks. Cette commande est exécutée sur le nœud de contrôle et le nom du playbook à exécuter est transmis en tant qu'argument :

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

Lorsque vous exécutez le playbook, la sortie est générée de manière à afficher le play et les tâches en cours d'exécution. La sortie indique également les résultats de chaque tâche exécutée.

L'exemple suivant montre le contenu d'un playbook simple, puis le résultat de son exécution.

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

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

Notez que la valeur de la clé **name** pour chaque play et tâche est affichée lors de l'exécution du playbook. (La tâche **Gathering Facts** est une tâche particulière qui, généralement, est automatiquement exécutée par le module `setup` au début d'un play. Nous abordons cet aspect

dans la suite du cours.) Dans le cas de playbooks avec plusieurs plays et tâches, la définition des attributs **name** facilite la surveillance de la progression de l'exécution d'un playbook.

Vous devez également noter que la tâche **latest httpd version installed est changed** pour `servera.lab.example.com`. Cela signifie que la tâche a remplacé quelque chose sur cet hôte pour garantir le respect de la spécification. Dans ce cas, cela veut dire que le paquetage `httpd` n'a probablement pas été installé ou qu'il ne s'agissait pas de la dernière version.

En général, les tâches dans les playbooks Ansible sont idempotentes, vous pouvez donc exécuter le playbook plusieurs fois sans risque. Si l'état des hôtes gérés ciblés est déjà correct, vous ne devez apporter aucune modification. Par exemple, supposons que le playbook de l'exemple précédent est à nouveau exécuté :

```
[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
```

Cette fois, toutes les tâches ont réussi avec le statut **ok** et aucune modification n'a été signalée.

## Augmentation de la verbosité de sortie

La sortie par défaut fournie par la commande **ansible-playbook** ne fournit pas d'informations détaillées sur l'exécution de la tâche. La commande **ansible-playbook -v** fournit des informations supplémentaires, sur quatre niveaux maximum.

### Configuration de la verbosité de sortie de l'exécution du playbook

| OPTION       | DESCRIPTION  |
|--------------|--|
| <b>-v</b>    | Les résultats de la tâche sont affichés.   |
| <b>-vv</b>   | Les résultats et la configuration de la tâche sont affichés.   |
| <b>-vvv</b>  | Inclut des informations sur les connexions aux hôtes gérés.  |
| <b>-vvvv</b> | Ajoute des options de détails supplémentaires aux plug-ins de connexion, notamment les utilisateurs en cours actifs dans les hôtes gérés pour exécuter des scripts, et les scripts exécutés. |

## Vérification de la syntaxe

Avant d'exécuter un playbook, il est conseillé d'effectuer une vérification pour s'assurer que la syntaxe du contenu est correcte. La commande **ansible-playbook** propose une option **--syntax-check** que vous pouvez utiliser pour vérifier la syntaxe d'un playbook. L'exemple suivant illustre une vérification réussie de la syntaxe d'un playbook.

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

Une erreur de syntaxe est renvoyée en cas d'échec de la vérification. La sortie comprend également l'emplacement approximatif de l'erreur de syntaxe dans le playbook. L'exemple suivant illustre la vérification de l'échec de syntaxe d'un playbook. Comme vous pouvez le constater, il manque le séparateur d'espace après l'attribut **name** du play.

```
[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
```

## Exécution d'un essai à blanc

Vous pouvez utiliser l'option **-C** pour effectuer un *essai à blanc* de l'exécution du playbook. Lorsqu'il est exécuté, Ansible affiche les modifications qui seraient effectuées en cas d'exécution du playbook. Cependant, il n'apporte aucune modification réelle aux hôtes gérés.

L'exemple ci-dessous illustre l'essai à blanc d'un playbook contenant une seule tâche pour s'assurer que la version la plus récente du paquet *httpd* est bien installée sur un hôte géré. Notez que l'essai à blanc indique que la tâche apportera une modification sur l'hôte géré.

```
[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
```



## RÉFÉRENCES

Page de manuel (1)[ansible-playbook](#)

**Présentation des playbooks – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks\\_intro.html](https://docs.ansible.com/ansible/2.7/user_guide/playbooks_intro.html)

**Playbooks – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks.html](https://docs.ansible.com/ansible/2.7/user_guide/playbooks.html)

**Mode Check (« Essai à blanc ») – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks\\_checkmode.html](https://docs.ansible.com/ansible/2.7/user_guide/playbooks_checkmode.html)

## ► EXERCICE GUIDÉ

# ÉCRITURE ET EXÉCUTION DE PLAYBOOKS

Dans cet exercice, vous allez écrire et exécuter un playbook Ansible.

## RÉSULTATS

Vous devez pouvoir écrire un playbook à l'aide de la structure de playbook Ansible et de la syntaxe YAML de base, puis l'exécuter correctement avec la commande **ansible-playbook**.

Connectez-vous en tant qu'utilisateur student sur workstation, puis exécutez **lab playbook-basic setup**. Ce script de configuration garantit que les hôtes gérés, `serverc.lab.example.com` et `serverd.lab.example.com`, sont configurés pour l'atelier et sont accessibles sur le réseau. Il garantit également que le fichier de configuration Ansible et l'inventaire appropriés sont installés dans le répertoire de travail sur le nœud de contrôle.

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

Le répertoire de travail `/home/student/playbook-basic` a été créé sur workstation pour cet exercice. Ce répertoire contient déjà un fichier de configuration **ansible.cfg** et également un fichier d'inventaire **inventory**, ce dernier définissant le groupe `web` incluant les deux hôtes gérés listés ci-dessus en tant que membres.

Dans ce répertoire, utilisez un éditeur de texte pour créer un playbook nommé **site.yml**. Ce playbook contient un play qui doit cibler les membres du groupe d'hôtes `web`. Le playbook doit utiliser les tâches pour garantir le respect des conditions suivantes sur les hôtes gérés :

1. Le paquetage `httpd` est présent à l'aide du module `yum`.
2. Le fichier `files/index.html` local est copié dans `/var/www/html/index.html` sur chaque hôte géré à l'aide du module `copy`.
3. Le service `httpd` est démarré et activé à l'aide du module `service`.

La commande **ansible-doc** peut vous aider à comprendre les mots-clés nécessaires pour chacun des modules.

Une fois le playbook écrit, vérifiez sa syntaxe, puis exécutez-le à l'aide de **ansible-playbook** afin de mettre en œuvre la configuration.

- 1. Pour faciliter tous les exercices du playbook, vous pouvez utiliser l'éditeur de texte Vi pour modifier votre fichier `~/.vimrc` (créez-le si nécessaire) afin de vous assurer qu'il contient la ligne suivante :

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

Cela est facultatif, mais permet de configurer la commande **vi** de sorte que la clé **Tab** effectue automatiquement la mise en retrait à l'aide de deux espaces dans les fichiers YAML. Il vous est ainsi plus facile de modifier des playbooks Ansible.

► 2. Choisissez le répertoire **/home/student/playbook-basic**.

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

► 3. Utilisez un éditeur de texte pour créer un playbook appelé **/home/student/playbook-basic/site.yml**. Commencez à écrire un play ciblé sur les hôtes du groupe d'hôtes web.

- 3.1. Créez et ouvrez **~/playbook-basic/site.yml**. La première ligne du fichier doit contenir trois tirets pour indiquer le début du playbook.

```
---
```

- 3.2. La ligne suivante démarre le play. Elle doit commencer par un tiret et un espace avant le premier mot-clé du play. Nommez le play à l'aide d'une chaîne arbitraire décrivant l'objet du play, à l'aide du mot-clé **name**.

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

- 3.3. Ajoutez une paire mot-clé-valeur **hosts** pour spécifier que le play se déroule sur des hôtes dans le groupe d'hôtes web de l'inventaire. Assurez-vous que le mot-clé **hosts** mot-clé est en retrait de deux espaces afin de s'aligner sur le mot-clé **name** dans la ligne précédente.

Le fichier complet **site.yml** doit se présenter comme suit :

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

► 4. Continuez à modifier le fichier **/home/student/playbook-basic/site.yml** et ajoutez un mot-clé **tasks** et les trois tâches de votre play spécifiées dans les instructions.

- 4.1. Ajoutez un mot-clé **tasks** mis en retrait à l'aide de deux espaces (aligné sur le mot-clé **hosts**) pour démarrer la liste des tâches. Le fichier doit se présenter comme suit :

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

- 4.2. Ajoutez la première tâche. Mettez en retrait cette tâche à l'aide de quatre espaces, puis démarrez-la avec un tiret et un espace, puis nommez-la, par exemple, **httpd package is present**. Utilisez le module **yum** pour cette tâche. Mettez en retrait les mots-clés du module avec deux espaces supplémentaires ; définissez le nom du

paquetage sur **httpd** et l'état du paquetage sur **present**. La tâche doit se présenter comme suit :

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

- 4.3. Ajoutez la deuxième tâche. Calquez le format sur celui de la tâche précédente, puis nommez la tâche **correct index.html is present** par exemple. Utilisez le module **copy**. Les mots-clé du module doivent définir la clé **src** sur **files/index.html** et la clé **dest** sur **/var/www/html/index.html**. La tâche doit se présenter comme suit :

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

- 4.4. Ajoutez la troisième tâche pour lancer et activer le service **httpd**. Calquez le format sur celui de la tâche précédente, puis nommez la tâche **httpd is started** par exemple. Utilisez le module **service** pour cette tâche. Définissez la clé **name** du service sur **httpd**, l'attribut **state** sur **started** et la clé **enabled** sur **true**. La tâche doit se présenter comme suit :

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

- 4.5. Le playbook **site.yml** Ansible complet doit correspondre à l'exemple suivant. Assurez-vous que la mise en retrait des mots-clé de votre play, la liste des tâches et les mots-clés de chaque tâche sont corrects.

```
---
- 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
```

Enregistrez le fichier, puis quittez l'éditeur de texte.

- 5. Avant d'exécuter votre playbook, exécutez la commande **ansible-playbook --syntax-check site.yml** pour vérifier que sa syntaxe est correcte. En cas d'erreurs, corrigez-les avant de passer à l'étape suivante. Le résultat doit ressembler à ce qui suit :

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

- 6. Exécutez le playbook. Parcourez la sortie générée pour vous assurer que toutes les tâches ont été correctement exécutées.

```
[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
```

- 7. Le cas échéant, vous devez pouvoir exécuter à nouveau le playbook et constater que toutes les tâches sont terminées sans que les hôtes gérés n'aient été modifiés.

```
[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
```

- 8. Utilisez la commande **curl** pour vérifier que serverc et serverd sont tous deux configurés en tant que serveur HTTPD.

```
[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.
```

## Nettoyage

Sur workstation, exéutez le script **lab playbook-basic cleanup** pour effacer les ressources créées au cours de cet exercice.

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

L'exercice guidé est maintenant terminé.

# MISE EN ŒUVRE DE PLUSIEURS PLAYS

---

## OBJECTIFS

À la fin de cette section, les stagiaires seront en mesure d'effectuer les opérations suivantes :

- Écrire un playbook qui utilise plusieurs plays et l'augmentation des priviléges par play.
- Utiliser efficacement **ansible-doc** pour apprendre à utiliser de nouveaux modules visant à implémenter des tâches pour un play.

**Figure 3.0: Mise en œuvre de plusieurs plays**

## ÉCRITURE DE PLUSIEURS PLAYS

Un playbook est un fichier YAML contenant une liste d'un ou de plusieurs plays. Gardez à l'esprit qu'un seul play est une liste ordonnée de tâches à exécuter sur des hôtes sélectionnés de l'inventaire. Par conséquent, si un playbook contient plusieurs plays, chacun d'eux peut appliquer ses tâches à un ensemble distinct d'hôtes.

Cela peut s'avérer très utile lors de l'orchestration d'un déploiement complexe impliquant des tâches sur différents hôtes. Vous pouvez écrire un playbook pour exécuter un play sur un ensemble d'hôtes, et une fois cette opération terminée, un autre play traite un autre ensemble d'hôtes.

L'écriture d'un playbook contenant plusieurs plays est très simple. Chaque play du playbook est écrit comme un élément de liste de premier niveau. Chaque play est un élément de liste comprenant les mots-clés de play habituels.

L'exemple suivant illustre un playbook simple avec deux plays. Le premier play est exécuté sur `web.example.com` et le second sur `database.example.com`.

```
---
# 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
```

## UTILISATEURS DISTANTS ET AUGMENTATION DES PRIVILÈGES DANS LES PLAYS

Les plays peuvent utiliser d'autres paramètres d'utilisateurs distants ou d'augmentation des privilèges que ceux spécifiés par les paramètres par défaut du fichier de configuration. Ces paramètres sont définis dans chaque play au même niveau que les mots-clés **hosts** ou **tasks**.

### Attributs de l'utilisateur

En règle générale, les tâches des playbooks sont exécutées via une connexion réseau aux hôtes gérés. Comme c'est le cas pour les commandes ad hoc, le compte utilisateur utilisé pour l'exécution de ces tâches dépend de divers mots-clés du fichier de configuration Ansible, **/etc/ansible/ansible.cfg**. Vous pouvez définir l'utilisateur exécutant les tâches à l'aide du mot-clé **remote\_user**. Cependant, si l'augmentation des privilèges est activée, d'autres mots-clés tels que **become\_user** peuvent également jouer un rôle.

Si l'utilisateur distant défini dans la configuration Ansible relative à l'exécution des tâches ne convient pas, il peut être remplacé à l'aide du mot-clé **remote\_user** dans un play.

```
remote_user: remoteuser
```

### Attributs d'augmentation des privilèges

D'autres mots-clés sont également disponibles pour définir des paramètres d'augmentation des privilèges depuis un playbook. Vous pouvez utiliser le mot-clé booléen **become** pour activer ou désactiver l'augmentation des privilèges, indépendamment de la manière dont cette fonctionnalité est définie dans le fichier de configuration Ansible. **yes** ou **true** permet d'activer l'augmentation des privilèges et **no** ou **false** permet de la désactiver.

```
become: true
```

Si l'augmentation des privilèges est activée, vous pouvez utiliser le mot-clé **become\_method** pour définir la méthode d'augmentation des privilèges à appliquer dans le cadre d'un play spécifique. L'exemple ci-dessous indique que vous devez utiliser **sudo** pour l'augmentation des privilèges.

```
become_method: sudo
```

De plus, lorsque l'augmentation des privilèges est activée, le mot-clé **become\_user** peut définir le compte utilisateur à utiliser pour un play spécifique.

```
become_user: privileged_user
```

L'exemple suivant illustre l'utilisation de ces mots-clé dans un 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

```

## RECHERCHE DE MODULES POUR DES TÂCHES

### Documentation du module

Ansible est fourni avec un grand nombre de modules. Les administrateurs disposent ainsi de nombreux outils pour les tâches administratives courantes. Nous avons abordé, précédemment dans ce cours, le site Web de la documentation Ansible à l'adresse <http://docs.ansible.com>.

L'*index des modules* sur le site Web est un moyen facile de parcourir la liste des modules fournis avec Ansible. Par exemple, les modules destinés à la gestion des utilisateurs et des services sont situés sous *Systems Modules*, tandis que ceux relatifs à l'administration de base de données sont accessibles sous *Database Modules*.

Pour chaque module, le site Web de documentation Ansible propose un résumé des fonctions disponibles, ainsi que des instructions concernant l'appel de chacune d'elles, avec des options. La documentation fournit également des exemples utiles vous montrant comment utiliser chaque module et définir leurs mots-clés dans une tâche.

Vous avez déjà utilisé la commande **ansible-doc** pour consulter des informations sur les modules installés sur le système local. Souvenez-vous que vous pouvez consulter la liste des modules disponibles sur un nœud de contrôle en exécutant la commande **ansible-doc -l**. Celle-ci affiche la liste des noms de modules, ainsi qu'un résumé de leurs fonctions.

```
[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
```

Utilisez la commande **ansible-doc [module name]** pour afficher la documentation détaillée relative à un module. À l'instar du site Web de documentation Ansible, cette commande fournit un résumé de la fonction du module, des informations sur ses différentes options, ainsi que des exemples. L'exemple ci-dessous illustre la documentation affichée pour le module yum.

```
[student@workstation modules]$ ansible-doc yum
> YUM      (/usr/lib/python2.7/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.

* 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: False]
    type: bool
    version_added: 2.7

...output omitted...

EXAMPLES:
- name: install the latest version of Apache
  yum:
    name: httpd
    state: latest

- name: ensure a list of packages installed
  yum:
    name: "{{ packages }}"
  vars:
    packages:
      - httpd
      - httpd-tools

- name: remove the Apache package
  yum:
    name: httpd
    state: absent

...output omitted...

```

La commande **ansible-doc** propose également une option **-s**, qui génère un exemple de résultat pouvant servir de modèle concernant la méthode d'utilisation d'un module spécifique dans un playbook. Ce résultat peut faire office de modèle de démarrage à inclure éventuellement dans un playbook afin de mettre en œuvre le module en vue de l'exécution de tâches. Les résultats comprennent des commentaires pour rappeler aux administrateurs comment utiliser chaque option. L'exemple ci-dessous illustre le résultat pour le module 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 ...
```

## Maintenance de modules

Ansible est fourni avec un large éventail de modules utilisables pour de nombreuses tâches. La communauté en amont est très active, et ces modules peuvent être à différents stades de développement. La documentation **ansible-doc** relative au module est censée spécifier les personnes chargées de la maintenance de ce dernier dans la communauté Ansible en amont, ainsi que son stade de développement. Ces informations figurent dans la section **METADATA** à la fin de la sortie de **ansible-doc** du module.

Le champ **status** fait état du niveau de développement du module :

- **stableinterface** : les mots-clés du module sont stables et tous les efforts seront déployés pour ne pas supprimer de mots-clés ni modifier leur signification.
- **preview** : le module a atteint le niveau de version préliminaire et peut être instable, ses mots-clés sont amenés à évoluer ou il peut avoir besoin de bibliothèques ou de services Web qui sont eux-mêmes sujets à des modifications incompatibles.
- **deprecated** : le module est obsolète et va disparaître des prochaines versions.
- **removed** : le module a été supprimé de la version, mais un stub existe à des fins de documentation pour aider les précédents utilisateurs à migrer vers les nouveaux modules.

**NOTE**

Le statut **stableinterface** indique uniquement qu'une interface de module est stable, sans toutefois évaluer la qualité du code du module.

Le champ **supported\_by** fait état de la personne qui assure la maintenance du module dans la communauté Ansible en amont. Les valeurs possibles sont les suivantes :

- **core** : la maintenance est assurée par les développeurs Ansible « principaux » en amont et est toujours intégrée à Ansible.
- **curated** : les modules sont soumis aux partenaires ou aux entreprises au sein de la communauté qui en assurent la maintenance. Les responsables de la maintenance de ces modules doivent surveiller chaque problème signalé ou extraire les demandes relatives au module. Les développeurs « principaux » en amont examinent les modifications proposées concernant ces modules après que les responsables de leur maintenance ont approuvé les modifications. Les contributeurs principaux s'assurent également que tous les problèmes identifiés dans ces modules en raison de modifications apportées au moteur Ansible sont corrigés. Ces modules sont actuellement inclus dans Ansible, mais ils sont susceptibles d'être fournis séparément à l'avenir.
- **community** : modules non pris en charge par les développeurs principaux en amont, les partenaires ou les entreprises, mais leur maintenance est intégralement assurée par la communauté Open Source globale. Les modules appartenant à cette catégorie sont entièrement utilisables, mais la résolution des problèmes repose sur le bon vouloir de la communauté. Ces modules sont actuellement inclus dans Ansible, mais ils sont susceptibles d'être fournis séparément à l'avenir.

La communauté Ansible en amont effectue le suivi des problèmes relatifs à Ansible et des modules intégrés à l'adresse <https://github.com/ansible/ansible/issues>.

Il peut arriver qu'un module ne permette pas d'effectuer une tâche que vous souhaitez réaliser. En tant qu'utilisateur final, vous pouvez également écrire vos propres modules ou obtenir des modules de tiers. Ansible recherche des modules personnalisés dans l'emplacement défini par la variable d'environnement **ANSIBLE\_LIBRARY** ou, si celle-ci n'est pas définie, par un mot-clé **library** dans le fichier de configuration Ansible en cours. Ansible recherche également des modules personnalisés dans le répertoire **./library** relatif au playbook en cours d'exécution.

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

Les informations sur l'écriture des modules ne font pas partie de ce cours. Vous trouverez la documentation afférente à l'adresse [https://docs.ansible.com/ansible/2.7/dev\\_guide/developing\\_modules.html](https://docs.ansible.com/ansible/2.7/dev_guide/developing_modules.html).

**IMPORTANT**

Utilisez la commande **ansible-doc** pour en savoir plus sur l'utilisation des modules pour vos tâches.

Si possible, évitez d'utiliser les modules `command`, `shell` et `raw` dans des playbooks, en dépit de leur apparente simplicité d'utilisation. Dans la mesure où ils utilisent des commandes arbitraires, il est très facile d'écrire des playbooks non idempotents avec ces modules.

Par exemple, la tâche suivante qui utilise le module `shell` n'est pas idempotente. Chaque fois qu'un play est exécuté, il réécrit `/etc/resolv.conf`, même s'il contient déjà la ligne **nameserver 192.0.2.1**.

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

Il existe de nombreuses façons d'écrire des tâches à l'aide du module `shell` de manière idempotente. Dans certains cas, la meilleure méthode consiste à effectuer ces modifications et à utiliser `shell`. Il peut s'avérer plus rapide d'utiliser **ansible-doc** pour découvrir le module `copy` et d'utiliser ce dernier pour obtenir l'effet souhaité.

Dans l'exemple suivant, le fichier `/etc/resolv.conf` n'est pas réécrit s'il comprend déjà le contenu approprié :

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

Le module `copy` permet de vérifier aisément si l'état a déjà été activé. Si tel est le cas, aucune modification n'est effectuée. Le module `shell` se caractérise par une grande souplesse d'utilisation. Toutefois, il convient de faire preuve d'une attention accrue pour s'assurer qu'il s'exécute de manière idempotente.

Les playbooks idempotents peuvent être exécutés à plusieurs reprises pour vérifier que les systèmes se trouvent dans un état spécifique, sans interrompre leur fonctionnement si cet état est actif.

## VARIATIONS DE LA SYNTAXE DES PLAYBOOKS

La dernière partie de ce chapitre est consacrée à certaines variations de la syntaxe YAML ou des playbooks Ansible que vous pouvez rencontrer.

### Commentaires YAML

Vous pouvez également utiliser des commentaires pour améliorer la lisibilité. Dans YAML, tout ce qui se trouve à droite du symbole dièse (#) est un commentaire. Si du contenu se trouve à gauche du commentaire, faites précéder le symbole dièse d'un espace.

```
# This is a YAML comment
```

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

## Chaînes YAML

Dans le langage YAML, les chaînes n'ont généralement pas besoin d'être placées entre guillemets, même si elles contiennent des espaces. Si vous le souhaitez, vous pouvez placer les chaînes entre guillemets droits simples ou doubles.

```
this is a string
```

```
'this is another string'
```

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

Pour écrire des chaînes de plusieurs lignes, vous disposez de deux méthodes. Vous pouvez utiliser la barre verticale (|) pour indiquer que les sauts de ligne à l'intérieur des chaînes doivent être conservés.

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

Vous pouvez également utiliser le caractère « plus grand que » (>) pour indiquer que les sauts de ligne doivent être convertis en espaces et que les espaces de début de lignes doivent être supprimés. Cette méthode est généralement utilisée pour scinder les chaînes longues au niveau des caractères d'espacement, de sorte qu'elles occupent plusieurs lignes afin de garantir une meilleure lisibilité.

```
fold_newlines: >
    This is
    a very long,
    long, long, long
    sentence.
```

## Dictionnaires YAML

Nous vous avons présenté des collections de paires clé-valeur écrites comme des blocs mis en retrait, comme dans l'exemple suivant :

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

Vous pouvez également écrire des dictionnaires au format de bloc en ligne placés entre accolades, comme dans l'exemple suivant :

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

Dans la plupart des cas, nous vous conseillons d'éviter d'utiliser ce format, car il est plus difficile à lire. Toutefois, il est plus couramment utilisé au moins dans un cas. L'utilisation des *rôles* sera abordé ultérieurement dans ce cours. Lorsqu'un playbook inclut une liste de rôles, il est plus courant d'utiliser cette syntaxe afin de faciliter la distinction entre les rôles inclus dans un play et les variables transmises à un rôle.

## Listes YAML

Nous vous avons également présenté des listes écrites avec la syntaxe à un seul tiret :

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

Les listes respectent également un format en ligne délimité par des crochets, comme suit :

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

Évitez cette syntaxe car elle est généralement moins lisible.

## Écriture abrégée obsolète d'un playbook avec des paires clé=valeur

Certains playbooks peuvent utiliser une ancienne méthode d'écriture abrégée pour définir des tâches qui consiste à placer les paires clé-valeur du module sur la même ligne que le nom du module. Par exemple, vous pouvez rencontrer ce type de syntaxe :

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

Généralement, vous écrivez plutôt cette tâche comme suit :

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

De plus, vous devez utiliser la forme normale au lieu de la forme abrégée.

La forme normale contient davantage de lignes, mais elle facilite le travail. Les mots-clés de la tâche sont empilés verticalement et sont plus faciles à différencier. Vos yeux peuvent lire le play directement de haut en bas, ce qui limite les mouvements oculaires de gauche à droite. En outre, contrairement à l'écriture abrégée, la syntaxe normale utilise le langage YAML natif. Les outils de mise en surbrillance de la syntaxe dans les éditeurs de texte modernes sont plus efficaces avec le format normal que le format abrégé.

Vous pouvez être amené à rencontrer cette syntaxe dans des manuels et des playbooks plus anciens écrits par d'autres personnes, et elle fonctionne encore.



## RÉFÉRENCES

Pages de manuel **ansible-playbook(1)** et **ansible-doc(1)**

**Présentation des playbooks – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks\\_intro.html](https://docs.ansible.com/ansible/2.7/user_guide/playbooks_intro.html)

**Playbooks – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks.html](https://docs.ansible.com/ansible/2.7/user_guide/playbooks.html)

**Développement de modules – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/dev\\_guide/developing\\_modules.html](https://docs.ansible.com/ansible/2.7/dev_guide/developing_modules.html)

**Prise en charge des modules – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/user\\_guide/modules\\_support.html](https://docs.ansible.com/ansible/2.7/user_guide/modules_support.html)

## ► EXERCICE GUIDÉ

# MISE EN ŒUVRE DE PLUSIEURS PLAYS

Au cours de cet exercice, vous allez écrire et utiliser un playbook contenant plusieurs plays, et l'utiliser pour effectuer des tâches administratives sur des hôtes gérés.

## RÉSULTATS

Vous devez pouvoir créer et exécuter un playbook afin de gérer la configuration et d'effectuer des tâches administratives sur un hôte géré.

Connectez-vous en tant qu'utilisateur **student** sur **workstation**, puis exécutez **lab playbook-multi setup**. Ce script de configuration garantit que l'hôte géré, **servera.lab.example.com**, est accessible sur le réseau. Il garantit également que le fichier de configuration Ansible et le fichier d'inventaire appropriés sont installés sur le noeud de contrôle.

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

Un développeur en charge du site intranet de votre entreprise vous a demandé d'écrire un playbook afin d'automatiser la configuration de l'environnement serveur sur **servera.lab.example.com**.

Un répertoire de travail, **/home/student/playbook-multi**, a été créé sur **workstation** pour le projet Ansible. Le répertoire contient déjà un fichier de configuration **ansible.cfg** et un fichier d'inventaire, **inventory**. L'hôte géré, **servera.lab.example.com**, est déjà défini dans ce dossier d'inventaire.

Dans ce répertoire, créez un playbook nommé **intranet.yml** contenant deux plays. Le premier play exige une augmentation des privilèges et doit effectuer les tâches suivantes dans l'ordre spécifié :

1. Utilisez le module **yum** pour vous assurer que les versions les plus récentes des paquetages **httpd** et **firewalld** sont installées.
2. Assurez-vous que le service **firewalld** est activé et démarré.
3. Vérifiez que **firewalld** est configuré pour autoriser les connexions au service **httpd**.
4. Assurez-vous que le service **httpd** est activé et démarré.
5. Veillez à ce que le fichier **/var/www/html/index.html** de l'hôte géré contienne "**Welcome to the example.com intranet!**".

Le second play n'exige pas d'augmentation des privilèges et doit exécuter une seule tâche à l'aide du module **uri** pour confirmer que l'URL **http://servera.lab.example.com** renvoie le code de statut HTTP **200**. Pour valider le contenu du serveur Web, configurez le module **uri** pour renvoyer le contenu de la requête Web dans les résultats de la tâche.

Selon les pratiques recommandées, les plays et les tâches doivent porter un nom qui renseigne sur leur objet, mais cela reste facultatif. L'exemple de solution nomme les plays et les tâches.

Gardez à l'esprit que vous pouvez utiliser la commande **ansible-doc** pour obtenir de l'aide sur vos recherches et l'utilisation de modules pour vos tâches.

Une fois que vous avez écrit le playbook, vérifiez sa syntaxe, puis exécutez-le afin de configurer et de tester le serveur Web. Utilisez l'option **-v** avec la commande **ansible-playbook** pour afficher la réponse du serveur Web. Vérifiez que la réponse du serveur Web correspond à la valeur attendue de « **Welcome to the example.com intranet!\n** ».

- ▶ 1. Modifiez le répertoire de travail **/home/student/playbook-multi**.

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

- ▶ 2. Créez un playbook, **/home/student/playbook-multi/intranet.yml**, et ajoutez-y les lignes nécessaires pour démarrer le premier play. Il doit cibler l'hôte géré **servera.lab.example.com** et activer l'augmentation des priviléges.
- 2.1. Créez le playbook **/home/student/playbook-multi/intranet.yml** et ouvrez-le. Ajoutez ensuite une ligne commençant par trois tirets pour indiquer le début du fichier YAML.

```
---
```

- 2.2. Ajoutez la ligne suivante au fichier **/home/student/playbook-multi/intranet.yml** pour marquer le début d'un play avec le nom **Enable intranet services**.

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

- 2.3. Ajoutez la ligne suivante pour indiquer que le play s'applique à l'hôte géré **servera.lab.example.com**. Veillez à mettre en retrait la ligne à l'aide de deux espaces (en l'alignant sur le mot-clé **name** au-dessus) pour indiquer qu'elle fait partie du premier play.

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

- 2.4. Ajoutez la ligne suivante pour activer l'augmentation des priviléges. Veillez à mettre en retrait la ligne à l'aide de deux espaces (en l'alignant sur les mots-clés au-dessus) pour indiquer qu'elle fait partie du premier play.

```
become: yes
```

- ▶ 3. Ajoutez la ligne suivante pour définir le début de la liste **tasks**. Mettez en retrait la ligne à l'aide de deux espaces (en l'alignant sur les mots-clés au-dessus) pour indiquer qu'elle fait partie du premier play.

```
tasks:
```

- 4. Comme il s'agit de la première tâche du premier play, définissez une tâche permettant de s'assurer que les paquetages *httpd* et *firewalld* sont à jour.

- 4.1. Sous le mot-clé **tasks** du premier play, ajoutez les lignes suivantes au fichier */home/student/playbook-multi/intranet.yml*. Cette opération crée la tâche qui vérifie que les versions les plus récentes des paquetages *vsftpd* et *firewalld* sont installées.

Veuillez à mettre en retrait la première ligne de la tâche à l'aide de quatre espaces, d'un tiret, puis d'un espace. Cela indique que la tâche est un élément de la liste **tasks** du premier play.

La première ligne attribue un nom descriptif à la tâche. La deuxième ligne est mise en retrait avec six espaces et appelle le module *yum*. La ligne suivante, qui est un mot-clé **name**, est mise en retrait à l'aide de huit espaces. Elle indique au module *yum* les paquetages devant être à jour. Le mot-clé **name** (différent du nom de la tâche) du module *yum* peut être composé d'une liste de paquetages mise en retrait à l'aide de dix espaces sur les deux lignes suivantes. Après la liste, le mot-clé **state** mis en retrait à l'aide de huit espaces indique au module *yum* que la dernière version des paquetages doit être installée.

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

- 5. Ajoutez une tâche à la liste du premier play qui permet de s'assurer que le contenu correct figure dans */var/www/html/index.html*.

- 5.1. Ajoutez les lignes suivantes au fichier */home/student/playbook-multi/intranet.yml* pour créer la tâche confirmant que le contenu du fichier */var/www/html/index.html* est correct. Veuillez à mettre la ligne en retrait à l'aide de quatre espaces suivis d'un tiret, puis d'un espace. Cela indique que la tâche est incluse dans le play et qu'il s'agit d'un élément de la liste **tasks**.

La première entrée attribue un nom descriptif à la tâche. La deuxième entrée est mise en retrait avec six espaces et appelle le module *copy*. Les autres entrées sont mises en retrait à l'aide de huit espaces et transmettent les arguments nécessaires pour s'assurer que le contenu de la page Web est correct.

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

- 6. Définissez deux tâches supplémentaires dans le play pour vérifier que le service *firewalld* est en cours d'exécution et va être lancé au démarrage, avant d'autoriser les connexions au service *http*.

- 6.1. Ajoutez les lignes suivantes au fichier */home/student/playbook-multi/intranet.yml* afin de créer la tâche chargée de s'assurer que le service *firewalld* est activé et en cours d'exécution. Veuillez à mettre la ligne en retrait à

l'aide de quatre espaces suivis d'un tiret, puis d'un espace. Cela indique que la tâche est incluse dans le play et qu'il s'agit d'un élément de la liste **tasks**.

La première entrée attribue un nom descriptif à la tâche. La deuxième entrée est mise en retrait avec huit espaces et appelle le module **service**. Les autres entrées sont mises en retrait à l'aide de dix espaces et transmettent les arguments nécessaires pour s'assurer que le service firewalld est activé et démarré.

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

- 6.2. Ajoutez les lignes suivantes au fichier **/home/student/playbook-multi/intranet.yml** afin de créer la tâche permettant de s'assurer que firewalld autorise les connexions HTTP sur des systèmes distants. Veillez à mettre la ligne en retrait à l'aide de quatre espaces suivis d'un tiret, puis d'un espace. Cela indique que la tâche est incluse dans le play et qu'il s'agit d'un élément de la liste **tasks**.

La première entrée attribue un nom descriptif à la tâche. La deuxième entrée est mise en retrait avec six espaces et appelle le module **firewalld**. Les autres entrées sont mises en retrait à l'aide de huit espaces et transmettent les arguments nécessaires pour s'assurer que les connexions HTTP à distance sont autorisées en permanence.

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

- ▶ 7. Ajoutez une dernière tâche à la liste du premier play qui permet de s'assurer que le service **httpd** est en cours d'exécution et qu'il va être lancé au démarrage.

- 7.1. Ajoutez les lignes suivantes au fichier **/home/student/playbook-multi/intranet.yml** afin de créer la tâche chargée de s'assurer que le service **httpd** est activé et en cours d'exécution. Veillez à mettre la ligne en retrait à l'aide de quatre espaces suivis d'un tiret, puis d'un espace. Cela indique que la tâche est incluse dans le play et qu'il s'agit d'un élément de la liste **tasks**.

La première entrée attribue un nom descriptif à la tâche. La deuxième entrée est mise en retrait avec six espaces et appelle le module **service**. Les autres entrées sont mises en retrait à l'aide de huit espaces et transmettent les arguments nécessaires pour s'assurer que le service **httpd** est activé et en cours d'exécution.

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

- 8. Dans **/home/student/playbook-multi/intranet.yml**, définissez un second play ciblé sur localhost qui va tester le serveur Web de l'intranet. Il n'exige pas d'augmentation des priviléges.

- 8.1. Ajoutez la ligne suivante au fichier **/home/student/playbook-multi/intranet.yml** pour marquer le début d'un second play.

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

- 8.2. Ajoutez la ligne suivante au fichier **/home/student/playbook-multi/intranet.yml** pour indiquer que le play s'applique à l'hôte géré **localhost**. Veillez à mettre la ligne en retrait avec deux espaces pour indiquer qu'elle est incluse dans le second play.

```
hosts: localhost
```

- 8.3. Ajoutez la ligne suivante au fichier **/home/student/playbook-multi/intranet.yml** pour désactiver l'augmentation des priviléges. Veillez à aligner la mise en retrait du mot-clé **become** sur le mot-clé **hosts** au-dessus.

```
become: no
```

- 9. Ajoutez la ligne suivante au fichier **/home/student/playbook-multi/intranet.yml** pour définir le début de la liste **tasks**. Veillez à mettre la ligne en retrait avec deux espaces pour indiquer qu'elle est incluse dans le second play.

```
tasks:
```

- 10. Ajoutez une seule tâche au deuxième play et utilisez le module **uri** pour demander du contenu à **http://servera.lab.example.com**. La tâche doit vérifier un code de statut HTTP de retour de **200**. Configurez la tâche pour placer le contenu renvoyé dans les résultats de la tâche.

- 10.1. Ajoutez les lignes suivantes au fichier **/home/student/playbook-multi/intranet.yml** pour créer la tâche de vérification des services Web à partir du nœud de contrôle. Veillez à mettre la première ligne en retrait à l'aide de quatre espaces suivis d'un tiret, puis d'un espace. Cela indique que la tâche est un élément de la liste **tasks** du second play.

La première ligne attribue un nom descriptif à la tâche. La deuxième ligne est mise en retrait avec six espaces et appelle le module **uri**. Les autres lignes sont mises en retrait à l'aide de huit espaces. Elles transmettent les arguments nécessaires pour exécuter une requête portant sur le contenu Web depuis le nœud de contrôle jusqu'à l'hôte géré et pour vérifier le code de statut reçu. Le mot-clé **return\_content** garantit que la réponse du serveur est ajoutée aux résultats de la tâche.

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

- 11. Vérifiez que le playbook **/home/student/playbook-multi/intranet.yml** final reflète le contenu structuré suivant.

```
---
- 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 http service
      firewalld:
        service: http
        permanent: true
        state: enabled
        immediate: yes

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

- 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
```

- 12. Enregistrez et fermez le fichier.

- 13. Exécutez la commande **ansible-playbook --syntax-check** pour vérifier la syntaxe du playbook **intranet.yml**.

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

- 14. Exécutez le playbook en utilisant le bouton **-v** possibilité de produire des résultats détaillés pour chaque tâche. Parcourez la sortie générée pour vous assurer que toutes les tâches ont été correctement exécutées. Vérifiez qu'une requête HTTP GET à `http://servera.lab.example.com` fournit le contenu correct.

```
[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"①: "Welcome to the example.com intranet!\n", "content_length": "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"②: 200, "url": "http://servera.lab.example.com"}  
  
PLAY RECAP ****  
localhost : ok=2     changed=0      unreachable=0    failed=0
```

```
servera.lab.example.com      : ok=6      changed=4      unreachable=0      failed=0
```

- ➊ Le serveur a répondu avec le contenu souhaité, **Welcome to the example.com intranet!**\n.
- ➋ Le serveur a répondu avec le code de statut HTTP **200**.

## Nettoyage

Sur **workstation**, exécutez le script **lab playbook-multi cleanup** pour effacer les ressources créées au cours de cet exercice.

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

L'exercice guidé est maintenant terminé.

## ► OPEN LAB

# MISE EN ŒUVRE DE PLAYBOOKS

## LISTE DE CONTRÔLE DES PERFORMANCES

Dans cet atelier, vous allez configurer des tâches administratives et les exécuter sur des hôtes gérés à l'aide d'un playbook.

## RÉSULTATS

Vous devez pouvoir créer et exécuter un playbook afin d'installer, de configurer et de vérifier le statut des services Web et de base de données sur un hôte géré.

Connectez-vous à `workstation` en tant qu'utilisateur `student` avec le mot de passe `student` et exécutez `lab playbook-review setup`. Ce script de configuration garantit que l'hôte géré, `serverb.lab.example.com`, est accessible sur le réseau. Il garantit également que le fichier de configuration Ansible et le fichier d'inventaire appropriés sont installés sur le noeud de contrôle.

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

Un développeur en charge du site Internet de l'entreprise vous a demandé d'écrire un playbook Ansible afin d'automatiser l'installation de son environnement serveur sur `serverb.lab.example.com`.

Un répertoire de travail, `/home/student/playbook-review`, a été créé sur `workstation` pour le projet Ansible. Le répertoire contient déjà le fichier de configuration `ansible.cfg` et le fichier `inventory`. L'hôte géré, `serverb.lab.example.com`, est déjà défini dans ce fichier d'inventaire.

Dans ce répertoire, créez un playbook nommé `internet.yml` contenant deux plays. Le premier play exige une augmentation de privilèges et doit effectuer les tâches suivantes dans l'ordre spécifié :

1. Utilisez le module `yum` pour vous assurer que les versions les plus récentes des paquetages suivants sont installées : `firewalld`, `httpd`, `php`, `php-mysql` et `mariadb-server`.
2. Assurez-vous que le service `firewalld` est activé et démarré.
3. Vérifiez que le service `firewalld` est configuré pour autoriser les connexions aux ports utilisés par le service `httpd`.
4. Assurez-vous que le service `httpd` est activé et démarré.
5. Assurez-vous que le service `mariadb` est activé et démarré.
6. Utilisez le module `get_url` pour vous assurer que le contenu de l'URL `http://materials.example.com/labs/playbook-review/index.php` a été installé en tant que fichier `/var/www/html/index.php` sur l'hôte géré.

Le second play n'exige pas d'augmentation des privilèges et doit exécuter une seule tâche à l'aide du module `uri` pour confirmer que l'URL `http://serverb.lab.example.com/` renvoie le code de statut HTTP 200.

Selon les pratiques recommandées, les plays et les tâches doivent porter un nom qui renseigne sur leur objet, mais cela reste facultatif. L'exemple de solution nomme les plays et les tâches.

Gardez à l'esprit que vous pouvez utiliser la commande **ansible-doc** pour obtenir de l'aide sur vos recherches et l'utilisation de modules pour vos tâches.

Une fois le playbook écrit, vérifiez sa syntaxe, puis exécutez-le afin de mettre en œuvre la configuration. Vérifiez votre travail en exécutant la commande **lab playbook-review grade**.



### NOTE

Le playbook utilisé dans cet atelier ressemble fortement à celui que vous avez écrit dans l'exercice guidé précédent dans ce chapitre. Si vous ne souhaitez pas créer intégralement le playbook de cet atelier, vous pouvez utiliser le playbook de cet exercice comme point de départ.

Si vous procédez de cette façon, veillez à cibler les hôtes corrects et à modifier les tâches afin qu'elles correspondent aux instructions de cet exercice.

1. Modifiez le répertoire de travail **/home/student/playbook-review**.
2. Créez un playbook, **/home/student/playbook-review/internet.yml**, et ajoutez les entrées nécessaires pour lancer un premier play nommé **Enable internet services**. Spécifiez ensuite l'hôte géré auquel il est destiné, `serverb.lab.example.com`. Ajoutez une entrée pour activer l'augmentation des priviléges.
3. Ajoutez les entrées nécessaires au fichier **/home/student/playbook-review/internet.yml** pour définir les tâches du premier play en vue de la configuration de l'hôte géré.
4. Dans **/home/student/playbook-review/internet.yml**, définissez un autre play pour que la tâche soit effectuée sur le nœud de contrôle afin de tester l'accès au serveur Web qui doit s'exécuter sur l'hôte géré `serverb`. L'augmentation des priviléges n'est pas nécessaire pour ce play.
5. Vérifiez la syntaxe du playbook **internet.yml** à l'aide de la commande **ansible-playbook**.
6. Exécutez le playbook avec la commande **ansible-playbook**. Parcourez la sortie générée pour vous assurer que toutes les tâches ont été correctement exécutées.

## Évaluation

Notez votre travail en exécutant la commande **lab playbook-review grade** à partir de votre poste de travail `workstation`. Corrigez toute erreur signalée et répétez le script tant que des erreurs persistent.

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

## Nettoyage

Sur `workstation`, exécutez le script **lab playbook-review cleanup** pour effacer les ressources créées au cours de cet atelier.

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

L'atelier est maintenant terminé.

## ► SOLUTION

# MISE EN ŒUVRE DE PLAYBOOKS

## LISTE DE CONTRÔLE DES PERFORMANCES

Dans cet atelier, vous allez configurer des tâches administratives et les exécuter sur des hôtes gérés à l'aide d'un playbook.

## RÉSULTATS

Vous devez pouvoir créer et exécuter un playbook afin d'installer, de configurer et de vérifier le statut des services Web et de base de données sur un hôte géré.

Connectez-vous à `workstation` en tant qu'utilisateur `student` avec le mot de passe `student` et exécutez `lab playbook-review setup`. Ce script de configuration garantit que l'hôte géré, `serverb.lab.example.com`, est accessible sur le réseau. Il garantit également que le fichier de configuration Ansible et le fichier d'inventaire appropriés sont installés sur le nœud de contrôle.

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

Un développeur en charge du site Internet de l'entreprise vous a demandé d'écrire un playbook Ansible afin d'automatiser l'installation de son environnement serveur sur `serverb.lab.example.com`.

Un répertoire de travail, `/home/student/playbook-review`, a été créé sur `workstation` pour le projet Ansible. Le répertoire contient déjà le fichier de configuration `ansible.cfg` et le fichier `inventory`. L'hôte géré, `serverb.lab.example.com`, est déjà défini dans ce fichier d'inventaire.

Dans ce répertoire, créez un playbook nommé `internet.yml` contenant deux plays. Le premier play exige une augmentation de privilèges et doit effectuer les tâches suivantes dans l'ordre spécifié :

1. Utilisez le module `yum` pour vous assurer que les versions les plus récentes des paquetages suivants sont installées : `firewalld`, `httpd`, `php`, `php-mysql` et `mariadb-server`.
2. Assurez-vous que le service `firewalld` est activé et démarré.
3. Vérifiez que le service `firewalld` est configuré pour autoriser les connexions aux ports utilisés par le service `httpd`.
4. Assurez-vous que le service `httpd` est activé et démarré.
5. Assurez-vous que le service `mariadb` est activé et démarré.
6. Utilisez le module `get_url` pour vous assurer que le contenu de l'URL `http://materials.example.com/labs/playbook-review/index.php` a été installé en tant que fichier `/var/www/html/index.php` sur l'hôte géré.

Le second play n'exige pas d'augmentation des privilèges et doit exécuter une seule tâche à l'aide du module `uri` pour confirmer que l'URL `http://serverb.lab.example.com/` renvoie le code de statut HTTP 200.

Selon les pratiques recommandées, les plays et les tâches doivent porter un nom qui renseigne sur leur objet, mais cela reste facultatif. L'exemple de solution nomme les plays et les tâches.

Gardez à l'esprit que vous pouvez utiliser la commande **ansible-doc** pour obtenir de l'aide sur vos recherches et l'utilisation de modules pour vos tâches.

Une fois le playbook écrit, vérifiez sa syntaxe, puis exécutez-le afin de mettre en œuvre la configuration. Vérifiez votre travail en exécutant la commande **lab playbook-review grade**.



### NOTE

Le playbook utilisé dans cet atelier ressemble fortement à celui que vous avez écrit dans l'exercice guidé précédent dans ce chapitre. Si vous ne souhaitez pas créer intégralement le playbook de cet atelier, vous pouvez utiliser le playbook de cet exercice comme point de départ.

Si vous procédez de cette façon, veillez à cibler les hôtes corrects et à modifier les tâches afin qu'elles correspondent aux instructions de cet exercice.

1. Modifiez le répertoire de travail **/home/student/playbook-review**.

```
[student@workstation ~] cd /home/student/playbook-review
```

2. Créez un playbook, **/home/student/playbook-review/internet.yml**, et ajoutez les entrées nécessaires pour lancer un premier play nommé **Enable internet services**. Spécifiez ensuite l'hôte géré auquel il est destiné, `serverb.lab.example.com`. Ajoutez une entrée pour activer l'augmentation des priviléges.

- 2.1. Ajoutez l'entrée suivante au début de **/home/student/playbook-review/internet.yml** pour marquer le début du format YAML.

```
---
```

- 2.2. Ajoutez l'entrée suivante pour marquer le début d'un play avec le nom **Enable internet services**.

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

- 2.3. Ajoutez l'entrée suivante pour indiquer que le play s'applique à l'hôte géré `serverb`. Assurez-vous de mettre en retrait l'entrée avec deux espaces.

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

- 2.4. Ajoutez l'entrée suivante pour activer l'augmentation des priviléges. Assurez-vous de mettre en retrait l'entrée avec deux espaces.

```
become: yes
```

3. Ajoutez les entrées nécessaires au fichier **/home/student/playbook-review/internet.yml** pour définir les tâches du premier play en vue de la configuration de l'hôte géré.

- 3.1. Ajoutez l'entrée suivante pour définir le début de la liste **tasks**. Assurez-vous de mettre en retrait l'entrée avec deux espaces.

```
tasks:
```

- 3.2. Ajoutez l'entrée suivante afin de créer une tâche chargée de vérifier l'installation des versions les plus récentes des paquetages nécessaires.

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

- 3.3. Ajoutez les entrées nécessaires au fichier **/home/student/playbook-review/internet.yml** pour définir les tâches de configuration du pare-feu.

```
- 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
```

- 3.4. Ajoutez les entrées nécessaires pour définir les tâches de gestion des services.

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

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

```
state: started
```

- 3.5. Ajoutez les entrées nécessaires afin de définir la tâche finale pour générer du contenu Web à des fins de test.

```
- 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
```

4. Dans **/home/student/playbook-review/internet.yml**, définissez un autre play pour que la tâche soit effectuée sur le nœud de contrôle afin de tester l'accès au serveur Web qui doit s'exécuter sur l'hôte géré **serverb**. L'augmentation des privilèges n'est pas nécessaire pour ce play.

- 4.1. Ajoutez l'entrée suivante pour marquer le début d'un second play nommé **Test internet web server**.

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

- 4.2. Ajoutez l'entrée suivante pour indiquer que le play s'applique à l'hôte géré **localhost**. Assurez-vous de mettre en retrait l'entrée avec deux espaces.

```
hosts: localhost
```

- 4.3. Ajoutez la ligne suivante après le mot-clé **hosts** pour désactiver l'augmentation des privilèges pour le second play. Assurez-vous de mettre en retrait l'entrée avec deux espaces.

```
become: no
```

- 4.4. Ajoutez une entrée au fichier **/home/student/playbook-review/internet.yml** pour définir le début de la liste **tasks**. Assurez-vous de mettre en retrait l'entrée avec deux espaces.

```
tasks:
```

- 4.5. Ajoutez les lignes suivantes pour créer la tâche de vérification des services Web de l'hôte géré à partir du nœud de contrôle.

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

5. Vérifiez la syntaxe du playbook **internet.yml** à l'aide de la commande **ansible-playbook**.

```
[student@workstation playbook-review]$ ansible-playbook --syntax-check \
```

```
> internet.yml  
playbook: internet.yml
```

6. Exécutez le playbook avec la commande **ansible-playbook**. Parcourez la sortie générée pour vous assurer que toutes les tâches ont été correctement exécutées.

```
[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
```

## Évaluation

Notez votre travail en exécutant la commande **lab playbook-review grade** à partir de votre poste de travail **workstation**. Corrigez toute erreur signalée et répétez le script tant que des erreurs persistent.

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

## Nettoyage

Sur workstation, exécutez le script **lab playbook-review cleanup** pour effacer les ressources créées au cours de cet atelier.

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

L'atelier est maintenant terminé.

# RÉSUMÉ

---

Dans ce chapitre, vous avez appris les principes suivants :

- Un *play* est une liste ordonnée de tâches qui doivent être exécutées sur des hôtes choisis dans l'inventaire.
- Un *playbook* est un fichier texte contenant une liste d'un ou plusieurs plays à exécuter dans l'ordre.
- Les playbooks Ansible sont écrits au format YAML.
- La structure des fichiers YAML adopte une mise en retrait avec des espaces afin de représenter la hiérarchie des données.
- Les tâches sont mises en œuvre à l'aide d'un code normalisé, fourni sous la forme de *modules* Ansible.
- La commande **ansible-doc** permet de lister les modules installés. Elle fournit également de la documentation et des exemples d'extraits de code concernant leur utilisation dans des playbooks.
- La commande **ansible-playbook** est utilisée pour vérifier la syntaxe des playbooks et exécuter ces derniers.



## CHAPITRE 4

# GESTION DES VARIABLES ET DES FAITS

### PROJET

Rédigez des cahiers qui utilisent des variables et des faits pour simplifier la gestion du cahier et des faits afin de référencer des informations sur les hôtes gérés.

### OBJECTIFS

- Créer et référencer des variables qui affectent des hôtes ou groupes d'hôtes particuliers, le play ou l'environnement global, et décrire le fonctionnement de la priorité des variables.
- Chiffrer les variables sensibles à l'aide d'Ansible Vault et exécutez des playbooks faisant référence à des fichiers de variables chiffrées par Vault.
- Référencer les données sur les hôtes gérés à l'aide de faits Ansible et configurer des faits personnalisés sur des hôtes gérés.

### SECTIONS

- Gestion des variables (et exercice guidé)
- Gérer les secrets (et exercice guidé)
- Gestion des faits (et exercice guidé)

### ATELIER

- Gestion des variables et des faits

# GESTION DES VARIABLES

---

## OBJECTIFS

Au terme de cette section, les stagiaires doivent pouvoir créer et référencer des variables dans des playbooks, qui affectent des hôtes ou groupes d'hôtes particuliers, le play ou l'environnement global, et décrire le fonctionnement de la priorité des variables.

## PRÉSENTATION DES VARIABLES ANSIBLE

Ansible prend en charge des variables qui permettent de stocker des valeurs en vue de leur réutilisation dans l'ensemble des fichiers au sein d'un projet Ansible. Ces variables simplifient la création et la maintenance d'un projet, et réduisent le nombre d'erreurs.

Elles facilitent également la gestion de valeurs dynamiques pour un environnement donné dans votre projet Ansible. Exemples de valeurs de variables :

- Utilisateurs à créer
- Paquets à installer
- Services à redémarrer
- Fichiers à supprimer
- Archives à récupérer sur Internet

## NOMS DES VARIABLES

Les noms de variables sont caractères composées de lettres, chiffres et traits de soulignement. Ils doivent commencer par une lettre.

Le tableau suivant illustre la différence entre les noms de variables valides et non valides.

### Exemples de noms de variables Ansible valides et non valides

| NOMS DE VARIABLES NON VALIDES | NOMS DE VARIABLES VALIDES         |
|-------------------------------|-----------------------------------|
| web server                    | web_server                        |
| remote.file                   | remote_file                       |
| 1st file                      | file_1<br>file1                   |
| remoteserver\$1               | remote_server_1<br>remote_server1 |

## DÉFINITION DE VARIABLES

Les variables peuvent être définies à des emplacements divers au sein d'un projet Ansible. Néanmoins, cela peut être simplifié à trois niveaux d'étendue de base :

- *Étendue globale* : variables définies à partir d'une ligne de commande ou d'une configuration Ansible
- *Étendue de play* : variables définies dans le play et les structures correspondantes
- *Étendue d'hôtes* : variables définies sur des groupes d'hôtes et des hôtes individuels par l'inventaire, une collecte de faits ou des tâches enregistrées

Si une même variable est définie sur plusieurs niveaux, le niveau avec la priorité la plus élevée l'emporte. Une étendue étroite a la priorité par rapport à une étendue plus vaste : les variables définies par l'inventaire sont remplacées par celles définies par le playbook, lesquelles sont remplacées par les variables définies sur la ligne de commande.

La hiérarchie des variables est abordée en détail dans la documentation Ansible vers laquelle un lien est disponible dans les Références figurant au bas de cette section.

## VARIABLES DE PLAYBOOKS

Les variables jouent un rôle important dans les playbooks Ansible car elles facilitent la gestion des données variables dans un playbook.

### Définition de variables dans des playbooks

Lorsque vous écrivez des playbooks, vous pouvez définir vos propres variables, puis invoquer ces valeurs dans une tâche. Par exemple, une variable nommée `web_package` peut être définie avec la valeur `httpd`. Une tâche peut ensuite appeler la variable à l'aide du module `yum` pour installer le paquetage `httpd`.

Les variables de playbook peuvent être définies de plusieurs manières. Une façon courante consiste à placer la variable dans un bloc **vars** au début d'un playbook :

```
- hosts: all
vars:
  user: joe
  home: /home/joe
```

Il est également possible de définir des variables de playbook dans des fichiers externes. Dans ce cas, au lieu d'utiliser un bloc **vars** dans le playbook, la directive **vars\_files** peut être utilisée, suivie d'une liste de noms de fichiers de variable externes correspondant à l'emplacement du playbook :

```
- hosts: all
vars_files:
  - vars/users.yml
```

Les variables de playbook sont alors définies dans ce ou ces fichiers au format YAML :

```
user: joe
home: /home/joe
```

## Utilisation de variables dans des playbooks

Une fois les variables déclarées, les administrateurs peuvent les utiliser dans des tâches. Pour désigner les variables, celles-ci sont placées entre doubles accolades ({{}}). Lorsque la tâche est exécutée, Ansible remplace la variable par sa valeur.

```
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 }}"
```



### IMPORTANT

Lorsqu'une variable est utilisée comme premier élément commençant une valeur, l'utilisation de guillemets est impérative. Cela empêche Ansible d'interpréter la référence de variable comme le début d'un dictionnaire YAML. Si les guillemets ne sont pas utilisés, le message suivant apparaît :

```
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 }}"
```

## VARIABLES D'HÔTE ET VARIABLES DE GROUPE

Les variables d'inventaire qui s'appliquent directement aux hôtes appartiennent à deux grandes catégories : les *variables d'hôtes*, qui s'appliquent à un hôte spécifique ; et les *variables de groupe*, qui s'appliquent à tous les hôtes d'un groupe d'hôtes ou d'un groupe de groupes d'hôtes. Les variables d'hôte l'emportent sur les variables de groupe, mais celles définies par un playbook l'emportent sur les deux précédentes.

La définition des variables d'hôte et des variables de groupe peut s'effectuer directement dans le fichier d'inventaire. Cette méthode, désormais obsolète, n'est pas privilégiée, mais il se peut que vous la rencontriez encore.

- Définition de la variable d'hôte `ansible_user` pour `demo.example.com`:

```
[servers]
```

```
demo.example.com ansible_user=joe
```

- Définition de la variable de groupe `user` pour le groupe d'hôtes `servers`.

```
[servers]
demo1.example.com
demo2.example.com

[servers:vars]
user=joe
```

- Définition de la variable de groupe `user` pour le groupe `servers`, qui se compose de deux groupes d'hôtes chacun avec deux serveurs.

```
[servers1]
demo1.example.com
demo2.example.com

[servers2]
demo3.example.com
demo4.example.com

[servers:children]
servers1
servers2

[servers:vars]
user=joe
```

Voici quelques quinconvénients de cette approche : elle rend l'utilisation de fichiers d'inventaire plus difficile, elle mélange des informations sur les hôtes et les variables au sein d'un même fichier et elle utilise une syntaxe obsolète.

## Utilisation des répertoires `group_vars` et `host_vars`

L'approche à privilégier pour définir des variables pour les hôtes et les groupes d'hôtes consiste à créer deux répertoires, **`group_vars`** et **`host_vars`**, dans le même répertoire de travail que le fichier d'inventaire ou le répertoire. Ces répertoires contiennent des fichiers qui définissent respectivement des variables de groupe et des variables d'hôtes.



### IMPORTANT

La méthode recommandée consiste à définir des variables d'inventaire à l'aide des répertoires **`host_vars`** et **`group_vars`**, et non à les définir directement dans les fichiers d'inventaire.

Pour définir des variables de groupe pour le groupe `servers`, vous devez créer un fichier YAML nommé **`group_vars/servers`**, puis son contenu définit des variables sur des valeurs en utilisant la même syntaxe que celle figurant dans un playbook :

```
user: joe
```

**CHAPITRE 4** | Gestion des variables et des faits

De même, pour définir des variables d'hôte pour un hôte donné, créez un fichier contenant les variables d'hôte et dont le nom correspond à l'hôte dans le répertoire **host\_vars**.

Les exemples suivants illustrent cette approche plus en détail. Prenons un scénario dans lequel deux datacenters doivent être gérés et où les hôtes des datacenters sont définis dans le fichier d'inventaire **~/project/inventory** :

```
[admin@station project]$ cat ~/project/inventory
[datacenter1]
demo1.example.com
demo2.example.com

[datacenter2]
demo3.example.com
demo4.example.com

[datacenters:children]
datacenter1
datacenter2
```

- Si vous devez définir une valeur générale pour tous les serveurs dans les deux datacenters, définissez une valeur pour le groupe d'hôtes **datacenters** :

```
[admin@station project]$ cat ~/project/group_vars/datacenters
package: httpd
```

- Si la valeur à définir varie d'un datacenter à l'autre, définissez une variable de groupe pour chaque groupe d'hôtes de datacenter :

```
[admin@station project]$ cat ~/project/group_vars/datacenter1
package: httpd
[admin@station project]$ cat ~/project/group_vars/datacenter2
package: apache
```

- Si la valeur à définir varie pour chaque hôte de chaque datacenter, définissez alors une variable dans des fichiers de variable d'hôte séparés :

```
[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
```

La structure de répertoire pour l'exemple de projet, **project**, s'il contenait tous les exemples de fichiers ci-dessus, apparaîtrait comme suit :

```
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

```

## REEMPLACEMENT DE VARIABLES À L'AIDE DE LA LIGNE DE COMMANDE

Les variables d'inventaire sont remplacées par les variables définies dans un playbook, mais ces deux types de variables peuvent être remplacés par des arguments transmis aux commandes **ansible** ou **ansible-playbook** sur la ligne de commande. Les variables définies sur la ligne de commande sont appelées des *variables supplémentaires*.

Des variables supplémentaires peuvent être utiles lorsque vous devez remplacer la valeur définie par une variable pour une exécution ponctuelle d'un playbook. Par exemple :

```
[user@demo ~]$ ansible-playbook main.yml -e "package=apache"
```

## VARIABLES ET MATRICES

Au lieu d'attribuer des données de configuration correspondant à cet élément (liste de paquetages, liste de services, liste d'utilisateurs, etc.) à plusieurs variables, les administrateurs peuvent utiliser des *matrices*. Les matrices présentent l'avantage de pouvoir être parcourues.

Par exemple, dans le snippet suivant :

```

user1_first_name: Bob
user1_last_name: Jones
user1_home_dir: /users/bjones
user2_first_name: Anne
user2_last_name: Cook
user3_home_dir: /users/acook

```

Cela peut être réécrit en utilisant une matrice nommée **users** :

```

users:
  bjones:
    first_name: Bob
    last_name: Jones
    home_dir: /users/bjones
  acook:
    first_name: Anne
    last_name: Cook
    home_dir: /users/acook

```

Vous pouvez ensuite utiliser les variables suivantes pour accéder aux données des utilisateurs :

```
# Returns 'Bob'
users.bjones.first_name

# Returns '/users/acook'
users.acook.home_dir
```

Parce que la variable est définie en tant que *dictionnaire* Python, une syntaxe alternative est disponible.

```
# Returns 'Bob'
users['bjones']['first_name']

# Returns '/users/acook'
users['acook']['home_dir']
```



### IMPORTANT

L'utilisation du point peut poser problème si les noms des clés sont identiques à ceux des attributs ou des méthodes Python, tels que **discard**, **copy**, **add** et autres. L'utilisation de crochets permet d'éviter les conflits et les erreurs.

Les deux syntaxes sont correctes, mais pour détecter plus facilement les erreurs, Red Hat recommande d'utiliser la même syntaxe sur l'ensemble des fichiers d'un même projet Ansible.

## VARIABLES ENREGISTRÉES

Les administrateurs peuvent utiliser l'instruction **register** pour capturer le résultat d'une commande. Le résultat est alors enregistré dans une variable qui pourra servir par la suite au débogage ou à d'autres fins, par exemple pour une configuration donnée liée au résultat d'une commande.

Le playbook suivant montre comment capturer la sortie d'une commande à des fins de débogage :

```
---
- 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
```

Lorsque vous exécutez le playbook, le module **debug** est utilisé pour envoyer la valeur de la variable enregistrée **install\_result** au terminal.

```
[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

```



## RÉFÉRENCES

### **Inventaire – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/user\\_guide/intro\\_inventory.html](https://docs.ansible.com/ansible/2.7/user_guide/intro_inventory.html)

### **Variables – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks\\_variables.html](https://docs.ansible.com/ansible/2.7/user_guide/playbooks_variables.html)

### **Ordre des variables : où placer une variable ?**

[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks\\_variables.html#variable-precedence-where-should-i-put-a-variable](https://docs.ansible.com/ansible/2.7/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable)

### **Syntaxe YAML – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/reference\\_appendices/YAMLSyntax.html](https://docs.ansible.com/ansible/2.7/reference_appendices/YAMLSyntax.html)

## ► EXERCICE GUIDÉ

# GESTION DES VARIABLES

Dans cet exercice, vous allez définir et utiliser des variables dans un playbook.

## RÉSULTATS

Vous devez pouvoir :

- Définir des variables dans un playbook.
- Créez des tâches qui incluent des variables définies.

Sur workstation, exécutez le script de configuration de l'atelier pour vérifier que l'environnement est prêt pour commencer l'exercice. Ce script crée le répertoire de travail **data-variables** et l'approvisionne à l'aide d'un fichier de configuration Ansible et d'un inventaire d'hôtes.

```
[student@workstation ~]$ lab data-variables setup
```

- 1. Sur workstation, en tant qu'utilisateur student, accédez au répertoire **~/data-variables**.

```
[student@workstation ~]$ cd ~/data-variables
[student@workstation data-variables]$
```

- 2. Dans les étapes suivantes, vous allez créer un playbook pour installer le serveur Web Apache et ouvrir les ports pour que le service soit joignable. Le playbook interroge le serveur Web pour vérifier qu'il est opérationnel.

Créez le playbook **playbook.yml** et définissez les variables suivantes dans la section **vars** :

### Variables

| VARIABLE         | DESCRIPTION  |
|------------------|--|
| web_pkg          | <b>Nom du paquetage à installer pour le serveur Web.</b> |
| firewall_pkg     | <b>Nom du paquetage de pare-feu.</b>                     |
| web_service      | <b>Nom du service Web à gérer.</b>                       |
| firewall_service | <b>Nom du service de pare-feu à gérer.</b>               |
| python_pkg       | <b>Nom du paquetage requis pour le module uri.</b>       |
| rule             | <b>Nom du service à ouvrir.</b>                          |

```
- name: Deploy and start Apache HTTPD service
hosts: webserver
vars:
  web_pkg: httpd
  firewall_pkg: firewalld
  web_service: httpd
  firewall_service: firewalld
  python_pkg: python-httplib2
  rule: http
```

- ▶ 3. Créez le bloc **tasks**, puis créez la première tâche qui doit utiliser le module `yum` pour s'assurer que les dernières versions des paquetages requis sont installées.

```
tasks:
- name: Required packages are installed and up to date
  yum:
    name:
      - "{{ web_pkg }}"
      - "{{ firewall_pkg }}"
      - "{{ python_pkg }}"
    state: latest
```



### NOTE

Vous pouvez utiliser `ansible-doc yum` pour passer en revue la syntaxe du module `yum`. La syntaxe indique que sa directive `name` peut prendre une liste des paquetages avec lesquels le module doit travailler, de sorte que vous n'avez pas besoin de tâches distinctes pour vérifier que chaque paquetage est à jour.

- ▶ 4. Créez deux tâches qui vont vérifier que les services `httpd` et `firewalld` sont démarrés et activés.

```
- 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
```

**NOTE**

Le module `service` fonctionne différemment du module `yum`, comme stipulé par **ansible-doc service**. Sa directive `name` prend le nom exact d'un service avec lequel travailler.

Vous pouvez écrire une seule tâche pour vous assurer que ces deux services sont démarrés et activés, en utilisant le mot-clé **loop** abordé plus tard dans ce cours.

- ▶ 5. Ajoutez une tâche qui garantit qu'un contenu spécifique existe dans le fichier `/var/www/html/index.html`.

```
- name: Web content is in place
copy:
  content: "Example web content"
  dest: /var/www/html/index.html
```

- ▶ 6. Ajoutez une tâche qui utilise le module `firewalld` pour s'assurer que les ports du pare-feu sont ouverts pour le service `firewalld` nommé dans la variable `rule`.

```
- name: The firewall port for {{ rule }} is open
firewalld:
  service: "{{ rule }}"
  permanent: true
  immediate: true
  state: enabled
```

- ▶ 7. Créez un nouveau play qui interroge le service Web pour vérifier que tout a été correctement configuré. Il sera exécuté sur `localhost`. De ce fait, Ansible n'a pas à changer d'identité, donc définissez le module `become` sur **false**. Vous pouvez utiliser le module `uri` pour vérifier une URL. Pour cette tâche, vérifiez que le code d'état 200 s'affiche, confirmant que le serveur Web sur `servera.lab.example.com` fonctionne et qu'il est correctement configuré.

```
- 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. Quand vous avez terminé, le playbook doit se présenter comme suit. Examinez le playbook et vérifiez que les deux plays sont corrects.

```
- name: Deploy and start Apache HTTPD service
hosts: webserver
vars:
```

```
web_pkg: httpd
firewall_pkg: firewalld
web_service: httpd
firewall_service: firewalld
python_pkg: python-httpplib2
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. Avant d'exécuter le playbook, utilisez la commande **ansible-playbook --syntax-check** pour vérifier sa syntaxe. En cas d'erreurs, corrigez-les avant de passer à l'étape suivante. Le résultat doit ressembler à ce qui suit :

```
[student@workstation data-variables]$ ansible-playbook --syntax-check playbook.yml
```

```
playbook: playbook.yml
```

- 10. Utilisez la commande **ansible-playbook** pour exécuter le playbook. Examinez le résultat pendant qu'Ansible installe les paquetages, qu'il démarre et active les services, et qu'il vérifie que le serveur Web est joignable.

```
[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
```

## Nettoyage

À partir de `workstation`, exécutez le script **lab data-variables cleanup** pour effacer l'exercice.

```
[student@workstation ~]$ lab data-variables cleanup
```

L'exercice guidé est maintenant terminé.

# GÉRER LES SECRETS

---

## OBJECTIFS

Au terme de cette section, les stagiaires doivent être en mesure de chiffrer les variables sensibles à l'aide d'Ansible Vault et d'exécuter des playbooks faisant référence à des fichiers de variables chiffrées par Vault.

## ANSIBLE VAULT

Il est possible qu'Ansible ait besoin d'accéder à des données sensibles telles que des mots de passe ou clés API pour configurer les hôtes gérés. Normalement, ces informations peuvent être stockées sous forme de texte clair dans les variables d'inventaire ou d'autres fichiers Ansible. Dans ce cas, toutefois, n'importe quel utilisateur ayant accès aux fichiers Ansible ou à un système de contrôle de version qui stocke les fichiers Ansible a accès à ces données sensibles. Cela pose un problème de sécurité évident.

Ansible Vault, fourni avec Ansible, peut être utilisé pour chiffrer et déchiffrer n'importe quel fichier de données structurées utilisé par Ansible. Pour utiliser Ansible Vault, un outil de ligne de commande appelé **ansible-vault** est utilisé pour créer, modifier, chiffrer, déchiffrer et afficher des fichiers. Ansible Vault peut chiffrer n'importe quel fichier de données structurées utilisé par Ansible. Cela peut inclure des variables d'inventaire, les fichiers de variables inclus dans un playbook, les fichiers de variables passés comme arguments lors de l'exécution du playbook ou les variables définies dans les rôles Ansible.



### IMPORTANT

Ansible Vault n'implémente pas ses propres fonctions cryptographiques, mais utilise plutôt une boîte à outils Python externe. Les fichiers sont protégés par un chiffrement symétrique AES256 avec un mot de passe comme clé secrète. Notez que la façon dont cela est réalisé n'a pas fait l'objet d'un audit formel de la part d'un tiers.

## Création d'un fichier chiffré

Pour créer un fichier chiffré, utilisez la commande **ansible-vault create filename**. La commande demande le nouveau mot de passe Vault, puis ouvre un fichier à l'aide de l'éditeur par défaut, **vi**. Vous pouvez définir et exporter la variable d'environnement **EDITOR** pour spécifier un éditeur par défaut différent en définissant et en exportant. Par exemple, pour définir **nano** comme éditeur par défaut, **export EDITOR=nano**.

```
[student@demo ~]$ ansible-vault create secret.yml
New Vault password: redhat
Confirm New Vault password: redhat
```

Au lieu d'entrer le mot de passe Vault par une saisie normale, vous pouvez utiliser un fichier de mot de passe Vault pour stocker le mot de passe Vault. Vous devez protéger ce fichier avec soin au moyen d'autorisations de fichiers, entre autres moyens.

```
[student@demo ~]$ ansible-vault create --vault-password-file=vault-pass secret.yml
```

Le chiffrement utilisé pour protéger les fichiers est au format AES256 dans les versions récentes d'Ansible, mais il est possible que les fichiers chiffrés par des versions plus anciennes utilisent toujours le format AES 128 bits.

## Consultation d'un fichier chiffré

Vous pouvez utiliser la commande **ansible-vault view filename** pour afficher un fichier chiffré Ansible Vault sans l'ouvrir pour le modifier.

```
[student@demo ~]$ ansible-vault view secret1.yml
Vault password: secret
less 458 (POSIX regular expressions)
Copyright (C) 1984-2012 Mark Nudelman

less comes with NO WARRANTY, to the extent permitted by law.
For information about the terms of redistribution,
see the file named README in the less distribution.
Homepage: http://www.greenwoodsoftware.com/less
my_secret: "yJJvPqhsiusmmPPZdnjndkdNjdg782meUZcw"
```

## Modification d'un fichier chiffré existant

Pour modifier un fichier chiffré existant, Ansible Vault propose la commande **ansible-vault edit filename**. Cette commande déchiffre le fichier dans un fichier temporaire et vous permet de le modifier. Lorsqu'il est enregistré, il copie le contenu et supprime le fichier temporaire.

```
[student@demo ~]$ ansible-vault edit secret.yml
Vault password: redhat
```



### NOTE

La sous-commande **edit** réécrit systématiquement le fichier, vous ne devez donc l'utiliser que pour effectuer des modifications. Cela peut avoir des conséquences en cas de contrôle de version du fichier. Vous devez toujours utiliser la sous-commande **view** pour afficher les contenus du fichier sans effectuer de modifications.

## Chiffrement d'un fichier existant

Pour chiffrer un fichier qui existe déjà, utilisez la commande **ansible-vault encrypt filename**. Cette commande peut prendre les noms de plusieurs fichiers pour les chiffrer en tant qu'arguments.

```
[student@demo ~]$ ansible-vault encrypt secret1.yml secret2.yml
New Vault password: redhat
Confirm New Vault password: redhat
Encryption successful
```

Utilisez l'option **--output=OUTPUT\_FILE** pour enregistrer le fichier chiffré sous un nouveau nom. Il n'est possible d'utiliser qu'un seul fichier d'entrée avec l'option **--output**.

## Déchiffrement d'un fichier existant

Un fichier chiffré existant peut être déchiffré de façon permanente à l'aide de la commande **ansible-vault decrypt filename**. Lors du déchiffrement d'un fichier unique, vous pouvez utiliser l'option **--output** pour enregistrer le fichier déchiffré sous un nom différent.

```
[student@demo ~]$ ansible-vault decrypt secret1.yml --output=secret1-decrypted.yml
Vault password: redhat
Decryption successful
```

## Modification du mot de passe d'un fichier chiffré

Vous pouvez utiliser la commande **ansible-vault rekey filename** pour modifier le mot de passe d'un fichier chiffré. Cette commande peut recomposer plusieurs fichiers de données en une fois. Elle demande le mot de passe d'origine, puis le nouveau mot de passe.

```
[student@demo ~]$ ansible-vault rekey secret.yml
Vault password: redhat
New Vault password: RedHat
Confirm New Vault password: RedHat
Rekey successful
```

Lorsqu'un fichier de mot de passe Vault est utilisé, faites appel à l'option **--new-vault-password-file**:

```
[student@demo ~]$ ansible-vault rekey \
> --new-vault-password-file=NEW_VAULT_PASSWORD_FILE secret.yml
```

## PLAYBOOKS ET ANSIBLE VAULT

Pour exécuter un playbook qui accède à des fichiers chiffrés avec Ansible Vault, vous devez fournir le mot de passe de chiffrement dans la commande **ansible-playbook**. Si vous ne fournissez pas le mot de passe, le playbook renvoie une erreur:

```
[student@demo ~]$ ansible-playbook site.yml
ERROR: A vault password must be specified to decrypt vars/api_key.yml
```

Pour fournir le mot de passe au playbook, utilisez l'option **--vault-id**. Par exemple, pour fournir le mot de passe Vault de manière interactive, utilisez **--vault-id @prompt** comme illustré dans l'exemple suivant :

```
[student@demo ~]$ ansible-playbook --vault-id @prompt site.yml
Vault password (default): redhat
```

**IMPORTANT**

Si vous utilisez une version d'Ansible antérieure à la version 2.4, vous devez utiliser l'option **--ask-vault-pass** pour fournir de manière interactive le mot de passe Vault. Vous pouvez toujours utiliser cette option si tous les fichiers chiffrés par Vault et utilisés par le playbook ont été chiffrés avec le même mot de passe.

```
[student@demo ~]$ ansible-playbook --ask-vault-pass site.yml
Vault password: redhat
```

Vous pouvez également utiliser l'option **--vault-password-file** pour spécifier un fichier qui stocke le mot de passe de chiffrement en texte clair. Le mot de passe doit être une chaîne stockée sous la forme d'une seule ligne dans le fichier. Étant donné que ce fichier contient le mot de passe en texte clair, il est essentiel de le protéger au moyen d'autorisations d'accès et d'autres mesures de sécurité.

```
[student@demo ~]$ ansible-playbook --vault-password-file=vault-pw-file site.yml
```

Vous pouvez également utiliser la variable d'environnement **ANSIBLE\_VAULT\_PASSWORD\_FILE** pour spécifier l'emplacement par défaut du fichier de mots de passe.

**IMPORTANT**

À partir d'Ansible 2.4, vous pouvez utiliser plusieurs mots de passe Ansible Vault avec **ansible-playbook**. Pour utiliser plusieurs mots de passe, transmettez plusieurs options **--vault-id** ou **--vault-password-file** à la commande **ansible-playbook**.

```
[student@demo ~]$ ansible-playbook \
> --vault-id one@prompt --vault-id two@prompt site.yml
Vault password (one):
Vault password (two):
...output omitted...
```

Les identifiants de Vault one et two qui précèdent @prompt peuvent correspondre à n'importe quel élément et vous pouvez même les omettre complètement. Si vous utilisez l'option **--vault-id id** lorsque vous chiffrerez un fichier avec la commande **ansible-vault**, cependant, lorsque vous exécutez **ansible-playbook** alors le mot de passe pour l'ID correspondant est essayé avant les autres. Si cela ne correspond pas, les autres mots de passe que vous avez fournis seront essayés ensuite. L'ID de coffre @prompt sans ID est en réalité une version abrégée de default@prompt indiquant de demander le mot de passe pour l'ID de Vault default.

## Pratiques recommandées en matière de gestion de fichier de variable

Pour simplifier la gestion, il est logique de configurer votre projet Ansible de sorte que les variables sensibles et toutes les autres soient conservées dans des fichiers distincts. Les fichiers contenant les variables sensibles peuvent ensuite être protégés à l'aide de la commande **ansible-vault**.

Pour rappel, la meilleure façon de gérer des variables d'hôte et de groupe consiste à créer des répertoires au niveau du playbook. Le répertoire **group\_vars** contient généralement des fichiers de variable dont les noms correspondent aux groupes d'hôtes auxquels ils s'appliquent. Le répertoire **host\_vars** contient généralement des fichiers de variable dont les noms correspondent aux noms des hôtes gérés auxquels ils s'appliquent.

Cependant, au lieu d'utiliser les fichiers dans **group\_vars** ou **host\_vars**, vous pouvez utiliser des répertoires pour chaque groupe d'hôtes ou hôte géré. Ces répertoires peuvent alors contenir plusieurs fichiers de variables, tous étant utilisés par le groupe d'hôtes ou l'hôte géré. Par exemple, dans le répertoire de projet suivant pour **playbook.yml**, les membres du groupe d'hôtes **webservers** utilisent les variables du fichier **group\_vars/webservers/vars**, et **demo.example.com** utilise les variables présentes à la fois dans **host\_vars/demo.example.com/vars** et **host\_vars/demo.example.com/vault** :

```
.  
├── ansible.cfg  
├── group_vars  
│   └── webservers  
│       └── vars  
├── host_vars  
│   └── demo.example.com  
│       ├── vars  
│       └── vault  
└── inventory  
└── playbook.yml
```

Ce scénario présente l'avantage de pouvoir placer la plupart des variables de **demo.example.com** dans le fichier **vars**, et les variables sensibles peut rester secrètes en les plaçant séparément dans le fichier **vault**. L'administrateur peut ensuite utiliser **ansible-vault** pour chiffrer **vault**, tout en conservant le fichier **vars** en texte clair.

Il n'y a rien de particulier à signaler concernant les noms de fichiers utilisés dans cet exemple dans le répertoire **host\_vars/demo.example.com**. Ce répertoire peut contenir davantage de fichiers, certains chiffrés par Ansible Vault et d'autres non.

Les variables de playbook (contrairement aux variables d'inventaire) peuvent également être protégées par Ansible Vault. Les variables de playbook sensibles peuvent être placées dans un fichier distinct, chiffré avec Ansible Vault et inclus dans le playbook via une directive **vars\_files**. Cela peut s'avérer utile, dans la mesure où les variables de playbook sont prioritaires sur les variables d'inventaire.

## Accélération des opérations Vault

Par défaut, Ansible utilise des fonctions du paquetage **python-crypto** pour chiffrer et déchiffrer les fichiers Vault. Si le nombre de fichiers chiffrés est important, le fait de les déchiffrer au démarrage peut entraîner un retard perceptible. Pour accélérer cette procédure, installez le paquet **python-cryptography**:

```
[student@demo ~]$ sudo yum install python-cryptography
```

Le paquet **python-cryptography** fournit une bibliothèque Python qui expose des primitives et des recettes cryptographiques. L'installation Ansible par défaut utilise PyCrypto pour ces opérations cryptographiques.

Si vous utilisez plusieurs mots de passe de Vault avec votre playbook, assurez-vous qu'un ID de Vault est attribué à chaque fichier chiffré et que vous entrez le mot de passe correspondant à cet ID de Vault lors de l'exécution du playbook. Cela garantit que le mot de passe correct est sélectionné en premier lors du déchiffrement du fichier chiffré par Vault, ce qui est plus rapide que d'obliger Ansible à essayer tous les mots de passe de Vault que vous avez fournis jusqu'à ce qu'il trouve le bon.

**Figure 4.0: Exécution avec Ansible Vault**



## RÉFÉRENCES

Pages de manuel **ansible-playbook(1)** et **ansible-vault(1)**

### **Vault – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks\\_vault.html](https://docs.ansible.com/ansible/2.7/user_guide/playbooks_vault.html)

### **Variables et coffres-forts – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks\\_best\\_practices.html#best-practices-for-variables-and-vaults](https://docs.ansible.com/ansible/2.7/user_guide/playbooks_best_practices.html#best-practices-for-variables-and-vaults)

## ► EXERCICE GUIDÉ

# GÉRER LES SECRETS

Dans cet exercice, vous allez chiffrer les variables sensibles avec Ansible Vault pour les protéger, puis exécuter un playbook utilisant ces variables.

## RÉSULTATS

Vous devez pouvoir réaliser les tâches suivantes :

- Utilisez les variables définies dans un fichier chiffré pour exécuter un playbook.

Connectez-vous à `workstation` en tant qu'utilisateur `student` avec le mot de passe `student`.

Sur `workstation`, exécutez le script `lab data-secret setup`. Ce script s'assure qu'Ansible est installé sur `workstation` et crée un répertoire de travail pour cet exercice. Ce répertoire comprend un fichier d'inventaire qui pointe vers `servera.lab.example.com` en tant qu'hôte géré, lequel fait partie du groupe `devservers`.

```
[student@workstation ~]$ lab data-secret setup
```

- 1. Sur `workstation`, en tant qu'utilisateur `student`, accédez au répertoire `~/data-secret`.

```
[student@workstation ~]$ cd ~/data-secret
```

- 2. Modifiez le contenu du fichier chiffré fourni, `secret.yml`. Le fichier peut être déchiffré en utilisant `redhat` comme mot de passe. Supprimez les commentaires des entrées de variables `username` et `pwhash`.

- 2.1. Modifiez le fichier chiffré `secret.yml` dans `~/data-secret`. Indiquez le mot de passe `redhat` pour le coffre lorsque vous y êtes invité. Le fichier crypté s'ouvre dans l'éditeur par défaut, `vim` .

```
[student@workstation data-secret]$ ansible-vault edit secret.yml
Vault password: redhat
```

- 2.2. Supprimez les commentaires des deux entrées de variables. Elles devraient apparaître comme suit :

```
username: ansibleuser1
pw: $6$jf...uxhP1
```

Enregistrez le fichier.

- 3. Créez un playbook qui utilise les variables définies dans le fichier chiffré **secret.yml**. Nommez le playbook **create\_users.yml** et créez-le dans le répertoire **~/data-secret**. Configurez le playbook de manière à utiliser le groupe d'hôtes **devservers**, lequel était défini dans le script de configuration d'atelier du fichier d'inventaire. Exécutez ce playbook avec l'identité **devops** sur l'hôte géré distant. Configurez le playbook pour créer l'utilisateur **ansibleuser1** défini par la variable **username**. Définissez le mot de passe de l'utilisateur à l'aide du hachage de mot de passe stocké dans la variable **pwhash**.

Le fichier **create\_users.yml** doit se présenter comme suit :

```
---
- 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. Utilisez la commande **ansible-playbook --syntax-check** pour vérifier la syntaxe du playbook **create\_users.yml**. Utilisez l'option **--ask-vault-pass** pour inviter à saisir le mot de passe Vault protégeant **secret.yml**. Résolvez les erreurs de syntaxe avant de continuer.

```
[student@workstation data-secret]$ ansible-playbook --syntax-check \
> --ask-vault-pass create_users.yml
Vault password (default): redhat

playbook: create_users.yml
```



#### NOTE

Au lieu d'utiliser **--ask-vault-pass**, vous pouvez utiliser l'option **--vault-id @prompt** la plus récente pour faire la même chose.

- 5. Créez un fichier de mot de passe à utiliser pour l'exécution du playbook au lieu de demander la saisie d'un mot de passe. Le fichier doit être nommé **vault-pass** et il doit stocker le mot de passe Vault **redhat** en texte clair. Remplacez les autorisations du fichier par **0600**.

```
[student@workstation data-secret]$ echo 'redhat' > vault-pass
[student@workstation data-secret]$ chmod 0600 vault-pass
```

- 6. Exécutez le playbook Ansible, en utilisant le fichier de mot de passe Vault afin de créer l'utilisateur **ansibleuser1** sur un système distant et en utilisant les mots de passe stockés

sous la forme de variables dans le fichier chiffré Ansible Vault **secret.yml**. Utilisez le fichier de mot de passe Vault **vault-pass**.

```
[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. Vérifiez que le playbook s'est exécuté correctement. L'utilisateur **ansibleuser1** devrait exister et avoir le mot de passe correct sur **servera.lab.example.com**. Testez ceci en utilisant **ssh** pour se connecter en tant que cet utilisateur sur **servera.lab.example.com**. Le mot de passe pour **ansibleuser1** est **redhat**. Pour s'assurer que **ssh** essaie seulement de s'authentifier par mot de passe et non par une clé SSH, utilisez l'option **-o PreferredAuthentications=password** lorsque vous vous connectez.

Déconnectez-vous de **servera** quand vous avez fini.

```
[student@workstation data-secret]$ ssh -o PreferredAuthentications=password \
> ansibleuser1@servera.lab.example.com
ansibleuser1@servera.lab.example.com's password: redhat
Warning: Permanently added 'servera.lab.example.com,172.25.250.10' (ECDSA) to the
list of known hosts.
[ansibleuser1@servera ~]$ exit
```

## Nettoyage

À partir de **workstation**, exécutez le script **lab data-secret cleanup** pour effacer l'exercice.

```
[student@workstation ~]$ lab data-secret cleanup
```

L'exercice guidé est maintenant terminé.

# GESTION DES FAITS

---

## OBJECTIFS

Au terme de cette section, les stagiaires doivent pouvoir référencer les données sur les hôtes gérés à l'aide de faits Ansible et configurer des faits personnalisés sur des hôtes gérés.

## FAITS ANSIBLE

Les faits *Ansible* sont des variables détectées automatiquement par Ansible sur un hôte géré. Les faits contiennent des informations propres à l'hôte qui peuvent être utilisées comme des variables régulières dans les plays, les conditions, les boucles ou toute autre instruction liée à une valeur collectée à partir d'un hôte géré.

Certains des faits rassemblés pour un hôte géré peuvent inclure :

- le nom de l'hôte ;
- la version du noyau ;
- les interfaces réseau ;
- les adresses IP ;
- la version du système d'exploitation ;
- différentes variables d'environnement ;
- le nombre d'UC ;
- la mémoire disponible ;
- l'espace disque disponible.

Les faits Ansible constituent un moyen pratique d'obtenir l'état d'un nœud géré et de décider de l'action à effectuer en fonction de cet état. Par exemple :

- Une tâche conditionnelle exécutée en fonction d'un fait contenant la version actuelle du noyau de l'hôte géré peut redémarrer un serveur.
- Vous pouvez personnaliser le fichier de configuration MySQL en fonction de la mémoire disponible signalée par un fait.
- Vous pouvez définir l'adresse IPv4 utilisée dans un fichier de configuration en fonction de la valeur d'un fait.

Généralement, chaque play exécute automatiquement le module `setup` avant la première tâche afin de rassembler les faits. Il s'agit de la tâche `Gathering Facts` dans Ansible 2.3 et version ultérieure, ou simplement de `setup` dans les versions antérieures d'Ansible. Par défaut, vous n'avez pas besoin d'avoir une tâche pour exécuter `setup` dans votre play. Il est normalement exécuté automatiquement pour vous.

Une façon de voir quels faits sont rassemblés pour vos hôtes gérés est de lancer un court playbook qui rassemble des faits et utilise le module `debug` pour imprimer la valeur de la variable `ansible_facts`.

```
- name: Fact dump
hosts: all
tasks:
  - name: Print all facts
    debug:
      var: ansible_facts
```

Lorsque vous exécutez le playbook, les faits sont affichés dans le résultat de la tâche :

```
[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...
  }
}
```

Le playbook affiche le contenu de la variable `ansible_facts` au format JSON en tant que hachage/dictionnaire de variables. Vous pouvez parcourir la sortie pour voir quels faits sont rassemblés, afin de trouver des faits que vous voudrez peut-être utiliser dans vos plays.

Le tableau suivant montre des faits collectés à partir d'un nœud géré et pouvant être utiles dans un playbook :

## Exemples de faits Ansible

| FAIT  | VARIABLE  |
|---|---|
| Nom d'hôte abrégé                                 | ansible_facts['hostname']                                     |
| Nom de domaine complet                            | ansible_facts['fqdn']   |
| Adresse IPv4 principale (en fonction du routage)  | ansible_facts['default_ipv4']['address']                      |
| Liste des noms de toutes les interfaces réseau    | ansible_facts['interfaces']                                   |
| Taille de la partition de disque <b>/dev/vda1</b> | ansible_facts['devices']['vda']['partitions']['vda1']['size'] |
| Liste des serveurs DNS                            | ansible_facts['dns']['nameservers']                           |
| Version du noyau en cours d'exécution             | ansible_facts['kernel']                                       |



### NOTE

N'oubliez pas que lorsque la valeur d'une variable est un hachage/dictionnaire, deux syntaxes peuvent être utilisées pour récupérer la valeur. Pour prendre deux exemples du tableau précédent :

- `ansible_facts['default_ipv4']['address']` peut aussi être écrit `ansible_facts.default_ipv4.address`
- `ansible_facts['dns']['nameservers']` peut aussi être écrit `ansible_facts.dns.nameservers`

Lorsqu'un fait est utilisé dans un playbook, Ansible remplace de manière dynamique le nom de la variable du fait par la valeur correspondante :

```
---
- hosts: all
  tasks:
    - name: Prints various Ansible facts
      debug:
        msg: >
          The default IPv4 address of {{ ansible_facts.fqdn }}
          is {{ ansible_facts.default_ipv4.address }}
```

La sortie suivante montre comment Ansible a interrogé le nœud géré et utilisé de manière dynamique les informations système pour mettre à jour la variable. Par ailleurs, les faits peuvent également être utilisés pour créer des groupes dynamiques d'hôtes qui correspondent à des critères particuliers.

```
[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

```

## FAITS ANSIBLE INJECTÉS SOUS FORME DE VARIABLES

Avant Ansible 2.5, les faits ont été injectés sous forme de variables individuelles précédées de la chaîne `ansible_` au lieu de faire partie de la variable `ansible_facts`. Par exemple, le fait `ansible_facts['distribution']` aurait été appelé `ansible_distribution`.

Beaucoup de playbooks plus anciens utilisent encore des faits injectés sous forme de variables au lieu de la nouvelle syntaxe avec un espace de noms sous la variable `ansible_facts`. Vous pouvez utiliser une commande ad hoc pour exécuter le module `setup` pour imprimer la valeur de tous les faits sous cette forme. Dans l'exemple suivant, une commande ad hoc est utilisée pour exécuter le module `setup` sur l'hôte géré `demo1.example.com`:

```

[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...
}

```

Le tableau suivant compare les anciens et les nouveaux noms de faits.

## Comparaison de certains noms de faits Ansible

| FORME ANSIBLE_FACTS   | ANCIENNE FORME DE VARIABLE DE FAIT                   |
|---|--|
| ansible_facts['hostname']                                     | ansible_hostname                                     |
| ansible_facts['fqdn']   | ansible_fqdn   |
| ansible_facts['default_ipv4']['address']                      | ansible_default_ipv4['address']                      |
| ansible['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                                       |



### IMPORTANT

Ansible reconnaît actuellement à la fois le nouveau système de nommage des faits (à l'aide de `ansible_facts`) et l'ancien système de nommage des « faits injectés en tant que variables séparées » des versions antérieures.

Vous pouvez désactiver l'ancien système de nommage en définissant le paramètre `inject_facts_as_vars` dans la section **[default]** du fichier de configuration Ansible sur **false**. Le paramètre par défaut est actuellement **true**.

La valeur par défaut de `inject_facts_as_vars` va probablement changer à **false** dans une future version d'Ansible. Si elle est définie sur **false**, vous ne pouvez référencer que des faits Ansibles en utilisant le nouveau système de nommage `ansible_facts.*`. Dans ce cas, toute tentative de référence à des faits via l'ancien espace de noms entraînera l'erreur suivante :

```
...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...
```

## DÉSACTIVATION DE LA COLLECTE DE FAITS

Parfois, vous ne voulez pas collecter les faits pour votre play. Plusieurs raisons peuvent justifier ce cas de figure. Il se peut que vous n'utilisiez aucun fait et que vous souhaitiez accélérer le play ou réduire la charge générée par le play sur les hôtes gérés. Il se peut que les hôtes gérés ne puissent pas exécuter le module `setup` pour une raison quelconque, ou que deviez installer un logiciel pré-requis avant de rassembler des faits.

Afin de désactiver la collecte des faits pour un play, définissez le mot-clé `gather_facts` sur **no** :

```
---
- name: This play gathers no facts automatically
  hosts: large_farm
  gather_facts: no
```

Même si `gather_facts: no` est défini pour un play, vous pouvez manuellement collecter les faits à tout moment en exécutant une tâche qui utilise le module `setup` :

```
tasks:
  - name: Manually gather facts
    setup:
      ...output omitted...
```

## FAITS PERSONNALISÉS

Les administrateurs peuvent créer des *faits personnalisés* qui sont stockés localement sur chaque hôte géré. Ces faits sont intégrés à la liste des faits standard collectés par le module `setup` lors de son exécution sur l'hôte géré. Ils permettent à l'hôte géré de fournir à Ansible des variables arbitraires, qui peuvent être utilisées pour ajuster le comportement des plays.

Des faits personnalisés peuvent être définis dans un fichier statique, formatés en tant que fichier INI ou en utilisant JSON. Il peut également s'agir de scripts exécutables qui génèrent une sortie JSON, exactement comme un script d'inventaire dynamique.

Les faits personnalisés permettent aux administrateurs de définir certaines valeurs pour les hôtes gérés dont les plays peuvent être utilisés pour remplir des fichiers de configuration ou exécuter des tâches de manière conditionnelle. Les faits personnalisés dynamiques permettent de déterminer par programmation les valeurs pour ces faits, ou même quels faits sont fournis quand le play est exécuté.

Par défaut, le module `setup` charge des faits personnalisés à partir de fichiers et de scripts du répertoire `/etc/ansible/facts.d` de chaque hôte géré. Le nom de chaque fichier ou script doit se terminer par **.fact** pour être utilisé. Les scripts de faits personnalisés dynamiques doivent produire des faits au format JSON et doivent être exécutables.

Voici un exemple de fichier de faits personnalisés statique écrit en format INI. Un fichier de faits personnalisés au format INI contient un niveau supérieur défini par une section, suivi par les paires clé-valeur des faits à définir :

```
[packages]
web_package = httpd
db_package = mariadb-server

[users]
user1 = joe
user2 = jane
```

Les mêmes faits peuvent être fournis au format JSON. Les faits JSON suivants sont équivalents aux faits spécifiés par le format INI dans l'exemple précédent. Les données JSON peuvent être stockées dans un fichier texte statique ou imprimées dans la sortie standard par un script exécutable :

```
{
  "packages": {
    "web_package": "httpd",
    "db_package": "mariadb-server"
  },
  "users": {
    "user1": "joe",
    "user2": "jane"
  }
}
```

**NOTE**

Les fichiers de faits personnalisés ne peuvent pas être au format YAML comme un playbook. Le format JSON est l'équivalent le plus proche.

Les faits personnalisés sont stockés par le module `setup` dans la variable `ansible_facts.ansible_local`. Les faits sont organisés en fonction du nom du fichier qui les définit. Par exemple, on peut supposer que les faits personnalisés précédents sont produits par un fichier enregistré sous `/etc/ansible/facts.d/custom.fact` sur l'hôte géré. Dans ce cas, la valeur de `ansible_facts.ansible_local['custom']['users']['user1']` est `joe`.

Vous pouvez vérifier la structure de vos faits personnalisés en exécutant le module `setup` sur les hôtes gérés avec une commande ad hoc.

```
[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...
  },
  "changed": false
}
```

Les faits personnalisés peuvent être utilisés de la même façon que les faits par défaut dans les playbooks :

```
[user@demo ~]$ cat playbook.yml
---
- hosts: all
```

```

tasks:
  - name: Prints various Ansible facts
    debug:
      msg: >
        The package to install on {{ ansible_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

```

## VARIABLES MAGIQUES

Certaines variables ne sont pas des faits, ou elles sont configurées par le biais du module `setup`, mais elles sont également définies automatiquement par Ansible. Ces *variables magiques* peuvent également s'avérer utiles pour obtenir des informations spécifiques sur un hôte géré particulier.

Parmi les quatre variables les plus utiles, on compte les suivantes :

### `hostvars`

Contient les variables pour les hôtes gérés et peut être utilisée afin d'obtenir les valeurs des variables d'un autre hôte géré. Cela n'inclut pas les faits de l'hôte géré s'ils n'ont pas encore été collectés pour cet hôte.

### `group_names`

Dresse la liste de tous les groupes dans lesquels se trouve l'hôte géré actuel.

### `groups`

Dresse la liste de tous les groupes et hôtes de l'inventaire.

### `inventory_hostname`

Contient le nom d'hôte de l'hôte géré actuel tel que configuré dans l'inventaire. Il peut, pour différentes raisons, être différent du nom d'hôte signalé par les faits.

Il existe également un certain nombre d'autres « variables magiques ». Pour plus d'informations, voir [https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks\\_variables.html#variable-precedence-where-should-i-put-a-variable](https://docs.ansible.com/ansible/2.7/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable) Un moyen d'obtenir leurs valeurs consiste à utiliser le module `debug` afin de consigner les contenus de la variable `hostvars` pour un hôte particulier :

```

[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": {}
}

```

```

"ansible_forks": 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"
}
}

```

Figure 4.0: Définition et utilisation de faits personnalisés Ansible



## RÉFÉRENCES

### **setup - Collecte de faits sur les hôtes distants – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/modules/setup\\_module.html](https://docs.ansible.com/ansible/2.7/modules/setup_module.html)

### **Faits locaux (Facts.d) – Variables – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks\\_variables.html#local-facts-facts-d](https://docs.ansible.com/ansible/2.7/user_guide/playbooks_variables.html#local-facts-facts-d)

## ► EXERCICE GUIDÉ

# GESTION DES FAITS

Dans cet exercice, vous allez collecter des faits Ansible à partir d'un hôte géré et les utiliser dans des plays.

## RÉSULTATS

Vous devez pouvoir :

- Collecter des faits à partir d'un hôte.
- Créer des tâches qui utilisent les faits collectés.

Sur **workstation**, exécutez le script de configuration de l'atelier pour vérifier que l'environnement est prêt pour commencer l'exercice. Ce script crée le répertoire de travail **data-facts** et l'approvisionne à l'aide d'un fichier de configuration Ansible et d'un inventaire d'hôtes.

```
[student@workstation ~]$ lab data-facts setup
```

- 1. Sur **workstation**, en tant qu'utilisateur **student**, accédez au répertoire **~/data-facts**.

```
[student@workstation ~]$ cd ~/data-facts
[student@workstation data-facts]$
```

- 2. Le module Ansible **setup** collecte des faits auprès de systèmes. Exécutez une commande ad hoc pour collecter des faits pour tous les serveurs dans le groupe **webserver**. Le résultat affiche tous les faits collectés pour **servera.lab.example.com** au format JSON. Passez en revue les variables affichées.

```
[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::5054:ff:fe00:fa0a"
        ],
    ...output omitted...
```

- 3. Sur **workstation**, créez un fichier de faits nommé **/home/student/data-facts/custom.fact**. Le fichier de faits définit le paquetage à installer et le service à démarrer sur **servera**. Le fichier doit se présenter comme suit :

```
[general]
package = httpd
service = httpd
state = started
```

- 4. Créez le playbook **setup\_facts.yml** pour définir **/etc/ansible/facts.d** comme répertoire à distance et enregistrer le fichier **custom факт** dans ce répertoire.

```
---
- 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. Exécutez une commande ad hoc à l'aide du module **setup**. Recherchez la section **ansible\_local** dans la sortie. À ce stade, aucun fait personnalisé ne doit apparaître.

```
[student@workstation data-facts]$ ansible webserver -m setup
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
...output omitted...
        "ansible_local": {}
...output omitted...
    },
    "changed": false
}
```

- 6. Avant d'exécuter le playbook, vérifiez que sa syntaxe est correcte à l'aide de la commande **ansible-playbook --syntax-check**. En cas d'erreurs, corrigez-les avant de passer à l'étape suivante. Le résultat doit ressembler à ce qui suit :

```
[student@workstation data-facts]$ ansible-playbook --syntax-check setup_facts.yml
playbook: setup_facts.yml
```

- 7. Exécutez le playbook **setup\_facts.yml**.

```
[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. Il est désormais possible de créer le playbook principal qui utilise les faits par défaut et les faits utilisateur pour configurer servera. Au cours des étapes suivantes, vous allez ajouter le fichier de playbook. Le fichier du playbook **playbook.yml** commence comme suit :

```
---
- name: Install Apache and starts the service
  hosts: webserver
```

- 9. Continuez à modifier le fichier **playbook.yml** en créant la première tâche, celle qui installe le paquetage *httpd*. Utilisez le fait d'utilisateur pour le nom du paquetage.

```
tasks:
  - name: Install the required package
    yum:
      name: "{{ ansible_facts.ansible_local.custom.general.package }}"
      state: latest
```

- 10. Créez la tâche qui utilise le fait personnalisé pour démarrer le service *httpd*.

```
  - name: Start the service
    service:
      name: "{{ ansible_facts.ansible_local.custom.general.service }}"
      state: "{{ ansible_facts.ansible_local.custom.general.state }}"
```

- 11. Une fois toutes les tâches terminées, le playbook complet doit se présenter comme suit : Examinez le playbook et vérifier que toutes les tâches sont définies.

```
---
- 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 }}"
```

- 12. Avant d'exécuter le playbook, utilisez une commande ad hoc pour vérifier que le service httpd n'est pas exécuté sur servera.

```
[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. Vérifiez la syntaxe du playbook en exécutant **ansible-playbook --syntax-check**. En cas d'erreurs, corrigez-les avant de passer à l'étape suivante. Le résultat doit ressembler à ce qui suit :

```
[student@workstation data-facts]$ ansible-playbook --syntax-check playbook.yml
playbook: playbook.yml
```

- 14. À l'aide de la commande **ansible-playbook**, exécutez le playbook. Examinez le résultat pendant qu'Ansible installe le paquetage, puis active le service.

```
[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. Utilisez une commande ad hoc pour exécuter **systemctl** afin de déterminer si le service httpd est en cours d'exécution sur servera.

```
[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; disabled; vendor preset:
  disabled)
```

```
Active: active (running) since Mon 2018-05-16 17:17:20 PDT; 12s ago
  Docs: man:httpd(8)
        man:apachectl(8)
Main PID: 32658 (httpd)
  Status: "Total requests: 0; Current requests/sec: 0; Current traffic:  0 B/
sec"
  CGroup: /system.slice/httpd.service
...output omitted...
```

## Nettoyage

À partir de `workstation`, exécutez le script `lab data-facts cleanup` pour effacer l'exercice.

```
[student@workstation ~]$ lab data-facts cleanup
```

L'exercice guidé est maintenant terminé.

## ► OPEN LAB

# GESTION DES VARIABLES ET DES FAITS

### LISTE DE CONTRÔLE DES PERFORMANCES

Dans cet atelier, vous allez écrire et exécuter un playbook Ansible qui utilise des variables, des secrets et des faits.

### RÉSULTATS

Vous devez pouvoir définir des variables et utiliser des faits dans un playbook, ainsi que des variables définies dans un fichier chiffré.

Connectez-vous en tant qu'utilisateur `student` sur `workstation`, puis exécutez `lab data-review setup`. Le script crée le répertoire de projet `data-review` et l'approvisionne à l'aide du fichier de configuration Ansible et de l'inventaire d'hôtes.

```
[student@workstation ~]$ lab data-review setup
```

Un développeur vous a demandé d'écrire un playbook Ansible pour automatiser la configuration d'un environnement de serveur Web sur `serverb.lab.example.com`, qui contrôle l'accès des utilisateurs à son site Web à l'aide de l'authentification de base.

Un répertoire de travail, `/home/student/data-review`, a été créé sur `workstation` pour ce projet. Le répertoire contient déjà un `ansible.cfg` et un fichier `inventory`. L'hôte géré, `serverb.lab.example.com`, est défini dans cette inventaire en tant que membre du groupe d'hôtes `webserver`.

Le sous-répertoire `files` contient un fichier de configuration `httpd.conf` qui configure le service Web Apache pour l'authentification de base. Le sous-répertoire contient également un fichier `.htaccess` pouvant être utilisé pour contrôler l'accès au répertoire racine des documents du serveur Web.

Dans le répertoire du projet, créez un playbook nommé `playbook.yml`, qui installe et configure le pare-feu et le service Web sur `serverb.lab.example.com`. Le playbook doit également créer un fichier `index.html` qui identifie le nom d'hôte et l'adresse IP de l'hôte géré à l'aide de faits Ansible. Ce fichier de contenu ne doit être visible que par les utilisateurs Web authentifiés avec succès. Pour maintenir la sécurité du mot de passe utilisé pour l'authentification de base, définissez le mot de passe de l'utilisateur en tant que variable dans un fichier chiffré par Ansible Vault.

Une fois le playbook créé, exécutez-le, puis vérifiez les résultats en récupérant le fichier de contenu à l'aide de l'authentification de base via HTTPS.

- Créez le playbook **playbook.yml**. Commencez le playbook en identifiant le groupe d'hôtes **webserver** en tant qu'hôte géré. Définissez les variables play suivantes :

### Variables

| VARIABLE                    | VALEURS                                   |
|-----------------------------|---|
| <code>firewall_pkg</code>   | <code>firewalld</code>                    |
| <code>web_pkg</code>        | <code>httpd</code>                        |
| <code>web_svc</code>        | <code>httpd</code>                        |
| <code>ssl_pkg</code>        | <code>mod_ssl</code>                      |
| <code>python_pkg</code>     | <code>python-passlib</code>               |
| <code>httpdconf_src</code>  | <code>files/httpd.conf</code>             |
| <code>httpdconf_file</code> | <code>/etc/httpd/conf/httpd.conf</code>   |
| <code>htaccess_src</code>   | <code>files/.htaccess</code>              |
| <code>secrets_dir</code>    | <code>/etc/httpd/secrets</code>           |
| <code>secrets_file</code>   | <code>"{{ secrets_dir }}/htpasswd"</code> |
| <code>web_root</code>       | <code>/var/www/html</code>                |
| <code>web_user</code>       | <code>guest</code>                        |

- Ajoutez une directive au play qui ajoute des variables supplémentaires à partir d'un fichier de variable nommé **vars/secret.yml**. Ce fichier contiendra une variable qui spécifie le mot de passe de l'utilisateur Web. Vous créerez ce fichier plus tard dans l'atelier.
- Ajoutez une section **tasks** au play. Écrivez une tâche qui garantit que la dernière version des paquetages nécessaires est installée. Ces paquetages sont définis par les variables `firewall_pkg`, `web_pkg`, `ssl_pkg` et `python_pkg`.
- Ajoutez une deuxième tâche au playbook pour vous assurer que le fichier spécifié par la variable `httpdconf_src` a été copié (avec le module `copy`) à l'emplacement spécifié par la variable `httpdconf_file` sur l'hôte géré. Le fichier doit appartenir à l'utilisateur `root` et au groupe `root`. Définissez également `0644` comme autorisations de fichier.
- Ajoutez une troisième tâche qui utilise le module `file` pour créer le répertoire spécifié par la variable `secrets_dir` sur l'hôte géré. Ce répertoire contient les fichiers de mots de passe utilisés pour l'authentification de base des services Web. Le fichier doit appartenir à l'utilisateur `apache` et au groupe `apache`. Définissez également `0500` comme autorisations de fichier.
- Ajoutez une quatrième tâche qui utilise le module `htpasswd` pour créer un fichier `htpasswd` à utiliser pour l'authentification de base des utilisateurs Web. L'attribut `path` doit être défini par la variable `secrets_file`. Le fichier doit contenir une entrée pour l'utilisateur défini par la variable `web_user`. Le mot de passe de l'utilisateur sera défini par la variable `web_pass` à ajouter plus tard. Le fichier doit appartenir à l'utilisateur `apache` et au groupe `apache`. Définissez également `0400` comme autorisations de fichier.

7. Ajoutez une cinquième tâche qui utilise le module `copy` pour créer un fichier `.htaccess` dans le répertoire racine du serveur Web. Copiez le fichier spécifié par la variable `htaccess_src` sur `{{web_root}}/.htaccess`. Le fichier doit appartenir à l'utilisateur apache et au groupe apache. Définissez 0400 comme autorisations de fichier.
8. Ajoutez une sixième tâche qui utilise le module `copy` pour créer le fichier de contenu Web `index.html` dans le répertoire spécifié par la variable `web_root`. Le fichier doit contenir le message « *NOM D'HÔTE (ADRESSE IP)* has been customized by Ansible. », où **NOM D'HÔTE** correspond au nom d'hôte complet de l'hôte géré et **ADRESSE IP** correspond à son adresse IP IPv4. Utilisez l'option `content` dans le module `copy` pour spécifier le contenu du fichier, et des faits Ansible pour spécifier le nom d'hôte et l'adresse IP.
9. Ajoutez une septième tâche utilisant le module `firewall` pour ouvrir sur le pare-feu le service `https` nécessaire aux utilisateurs afin d'accéder aux services Web sur l'hôte géré. Ce changement de pare-feu doit être permanent et doit avoir lieu immédiatement.
10. Ajoutez une huitième tâche utilisant le module `service` pour activer et redémarrer le service Web sur l'hôte géré pour que toutes les modifications de configuration prennent effet. Le nom du service Web est défini par la variable `web_svc`.
11. Ajoutez une tâche finale utilisant le module `service` pour activer et redémarrer le service de pare-feu sur l'hôte géré afin que toutes les modifications de configuration prennent effet.
12. Créez un fichier chiffré avec Ansible Vault, nommé `vars/secret.yml`. Il convient de définir la variable `web_pass` sur `redhat`, qui sera le mot de passe de l'utilisateur Web.
13. Exécutez le playbook `playbook.yml`.
14. Sur `workstation`, utilisez `curl` pour vérifier que le serveur Web est joignable via HTTPS. Vérifiez que l'authentification de base fonctionne en vous authentifiant en tant qu'utilisateur guest et en utilisant `redhat` comme mot de passe.

## Évaluation

Sur `workstation`, exécutez la commande `lab data-review grade` pour confirmer que l'exercice est réussi. Corrigez toute erreur signalée et répétez le script tant que des erreurs persistent.

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

## Nettoyage

À partir de `workstation`, exécutez le script `lab data-review cleanup` pour effacer l'exercice.

```
[student@workstation ~]$ lab data-review cleanup
```

L'atelier est maintenant terminé.

## ► SOLUTION

# GESTION DES VARIABLES ET DES FAITS

### LISTE DE CONTRÔLE DES PERFORMANCES

Dans cet atelier, vous allez écrire et exécuter un playbook Ansible qui utilise des variables, des secrets et des faits.

### RÉSULTATS

Vous devez pouvoir définir des variables et utiliser des faits dans un playbook, ainsi que des variables définies dans un fichier chiffré.

Connectez-vous en tant qu'utilisateur **student** sur **workstation**, puis exécutez **lab data-review setup**. Le script crée le répertoire de projet **data-review** et l'approvisionne à l'aide du fichier de configuration Ansible et de l'inventaire d'hôtes.

```
[student@workstation ~]$ lab data-review setup
```

Un développeur vous a demandé d'écrire un playbook Ansible pour automatiser la configuration d'un environnement de serveur Web sur **serverb.lab.example.com**, qui contrôle l'accès des utilisateurs à son site Web à l'aide de l'authentification de base.

Un répertoire de travail, **/home/student/data-review**, a été créé sur **workstation** pour ce projet. Le répertoire contient déjà un **ansible.cfg** et un fichier **inventory**. L'hôte géré, **serverb.lab.example.com**, est défini dans cette inventaire en tant que membre du groupe d'hôtes **webserver**.

Le sous-répertoire **files** contient un fichier de configuration **httpd.conf** qui configure le service Web Apache pour l'authentification de base. Le sous-répertoire contient également un fichier **.htaccess** pouvant être utilisé pour contrôler l'accès au répertoire racine des documents du serveur Web.

Dans le répertoire du projet, créez un playbook nommé **playbook.yml**, qui installe et configure le pare-feu et le service Web sur **serverb.lab.example.com**. Le playbook doit également créer un fichier **index.html** qui identifie le nom d'hôte et l'adresse IP de l'hôte géré à l'aide de faits Ansible. Ce fichier de contenu ne doit être visible que par les utilisateurs Web authentifiés avec succès. Pour maintenir la sécurité du mot de passe utilisé pour l'authentification de base, définissez le mot de passe de l'utilisateur en tant que variable dans un fichier chiffré par Ansible Vault.

Une fois le playbook créé, exécutez-le, puis vérifiez les résultats en récupérant le fichier de contenu à l'aide de l'authentification de base via HTTPS.

1. Créez le playbook **playbook.yml**. Commencez le playbook en identifiant le groupe d'hôtes **webserver** en tant qu'hôte géré. Définissez les variables play suivantes :

### Variables

| VARIABLE       | VALEURS                             |
|----------------|-------------------------------------|
| firewall_pkg   | <b>firewalld</b>                    |
| web_pkg        | <b>httpd</b>                        |
| web_svc        | <b>httpd</b>                        |
| ssl_pkg        | <b>mod_ssl</b>                      |
| python_pkg     | <b>python-passlib</b>               |
| httpdconf_src  | <b>files/httpd.conf</b>             |
| httpdconf_file | <b>/etc/httpd/conf/httpd.conf</b>   |
| htaccess_src   | <b>files/.htaccess</b>              |
| secrets_dir    | <b>/etc/httpd/secrets</b>           |
| secrets_file   | <b>"{{ secrets_dir }}/htpasswd"</b> |
| web_root       | <b>/var/www/html</b>                |
| web_user       | <b>guest</b>                        |

- 1.1. Accédez au répertoire de projet **data-review**.

```
[student@workstation ~]$ cd ~/data-review
[student@workstation data-review]$
```

- 1.2. Créez le fichier de playbook **playbook.yml** et modifiez-le dans un éditeur de texte.  
Le début du fichier doit se présenter comme suit :

```
---
- name: Install and configure webserver with basic auth
hosts: webserver
vars:
  firewall_pkg: firewalld
  web_pkg: httpd
  web_svc: httpd
  ssl_pkg: mod_ssl
  python_pkg: python-passlib
  httpdconf_src: files/httpd.conf
  httpdconf_file: /etc/httpd/conf/httpd.conf
  htaccess_src: files/.htaccess
  secrets_dir: /etc/httpd/secrets
  secrets_file: "{{ secrets_dir }}/htpasswd"
  web_root: /var/www/html
```

```
web_user: guest
```

2. Ajoutez une directive au play qui ajoute des variables supplémentaires à partir d'un fichier de variable nommé **vars/secret.yml**. Ce fichier contiendra une variable qui spécifie le mot de passe de l'utilisateur Web. Vous créerez ce fichier plus tard dans l'atelier.

À l'aide du mot-clé **vars\_files**, ajoutez les lignes suivantes au playbook pour indiquer à Ansible d'utiliser les variables trouvées dans le fichier de variable **vars/secret.yml**.

```
vars_files:
  - vars/secret.yml
```

3. Ajoutez une section **tasks** au play. Écrivez une tâche qui garantit que la dernière version des paquetages nécessaires est installée. Ces paquetages sont définis par les variables **firewall\_pkg**, **web\_pkg**, **ssl\_pkg** et **python\_pkg**.

3.1. Définissez le début de la section **tasks** en ajoutant la ligne suivante au playbook :

```
tasks:
```

- 3.2. Ajoutez les lignes suivantes au playbook pour définir une tâche utilisant le module **yum** pour installer les paquetages requis.

```
- name: Latest version of necessary packages installed
  yum:
    name:
      - "{{ firewall_pkg }}"
      - "{{ web_pkg }}"
      - "{{ ssl_pkg }}"
      - "{{ python_pkg }}"
    state: latest
```

4. Ajoutez une deuxième tâche au playbook pour vous assurer que le fichier spécifié par la variable **httpdconf\_src** a été copié (avec le module **copy**) à l'emplacement spécifié par la variable **httpdconf\_file** sur l'hôte géré. Le fichier doit appartenir à l'utilisateur **root** et au groupe **root**. Définissez également **0644** comme autorisations de fichier.

Ajoutez les lignes suivantes au playbook pour définir une tâche utilisant le module **copy** pour copier le contenu du fichier défini par la variable **httpdconf\_src** à l'emplacement spécifié par la variable **httpdconf\_file**.

```
- name: Configure web service
  copy:
    src: "{{ httpdconf_src }}"
    dest: "{{ httpdconf_file }}"
    owner: root
    group: root
    mode: 0644
```

5. Ajoutez une troisième tâche qui utilise le module **file** pour créer le répertoire spécifié par la variable **secrets\_dir** sur l'hôte géré. Ce répertoire contient les fichiers de mots de passe utilisés pour l'authentification de base des services Web. Le fichier doit appartenir à

l'utilisateur apache et au groupe apache. Définissez également 0500 comme autorisations de fichier.

Ajoutez les lignes suivantes au playbook pour définir une tâche utilisant le module `file` pour créer le répertoire défini par la variable `secrets_dir`.

```
- name: Secrets directory exists
  file:
    path: "{{ secrets_dir }}"
    state: directory
    owner: apache
    group: apache
    mode: 0500
```

6. Ajoutez une quatrième tâche qui utilise le module `htpasswd` pour créer un fichier `htpasswd` à utiliser pour l'authentification de base des utilisateurs Web. L'attribut `path` doit être défini par la variable `secrets_file`. Le fichier doit contenir une entrée pour l'utilisateur défini par la variable `web_user`. Le mot de passe de l'utilisateur sera défini par la variable `web_pass` à ajouter plus tard. Le fichier doit appartenir à l'utilisateur apache et au groupe **apache**. Définissez également **0400** comme autorisations de fichier.

Ajoutez les lignes suivantes au playbook pour définir une tâche utilisant le module `htpasswd` pour créer le fichier de mot de passe défini par la variable `secrets_file`. Définissez l'utilisateur à ajouter à l'aide de la variable `web_user`. Définissez le mot de passe de l'utilisateur à l'aide de la variable `web_pass`, qui sera définie ultérieurement dans un fichier chiffré.

```
- name: Web user exists in secrets file
  htpasswd:
    path: "{{ secrets_file }}"
    name: "{{ web_user }}"
    password: "{{ web_pass }}"
    owner: apache
    group: apache
    mode: 0400
```

7. Ajoutez une cinquième tâche qui utilise le module `copy` pour créer un fichier **.htaccess** dans le répertoire racine du serveur Web. Copiez le fichier spécifié par la variable `htaccess_src` sur `{{web_root}}/.htaccess`. Le fichier doit appartenir à l'utilisateur apache et au groupe apache. Définissez 0400 comme autorisations de fichier.

Ajoutez les lignes suivantes au playbook pour définir une tâche utilisant le module `copy` pour créer le fichier **.htaccess** au moyen du fichier défini par la variable `htaccess_src`.

```
- name: .htaccess file installed in docroot
  copy:
    src: "{{ htaccess_src }}"
    dest: "{{ web_root }}/.htaccess"
    owner: apache
    group: apache
    mode: 0400
```

8. Ajoutez une sixième tâche qui utilise le module `copy` pour créer le fichier de contenu Web **index.html** dans le répertoire spécifié par la variable `web_root`. Le fichier doit contenir le message « *NOM D'HÔTE (ADRESSE IP) has been customized by Ansible.* », où **NOM D'HÔTE** correspond au nom d'hôte complet de l'hôte géré et **ADRESSE IP** correspond à son adresse

IP IPv4. Utilisez l'option **content** dans le module `copy` pour spécifier le contenu du fichier, et des faits Ansible pour spécifier le nom d'hôte et l'adresse IP.

Ajoutez les lignes suivantes au playbook pour définir une tâche utilisant le module `copy` pour créer le fichier **index.html** dans le répertoire défini par la variable `web_root`. Remplissez le fichier avec le contenu spécifié à l'aide des faits Ansible `ansible_facts['fqdn']` et `ansible_facts['default_ipv4']['address']` extraits de l'hôte géré.

```
- 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"
```

9. Ajoutez une septième tâche utilisant le module `firewall` pour ouvrir sur le pare-feu le service `https` nécessaire aux utilisateurs afin d'accéder aux services Web sur l'hôte géré. Ce changement de pare-feu doit être permanent et doit avoir lieu immédiatement.

Ajoutez les lignes suivantes au playbook pour définir une tâche utilisant le module `firewall` afin d'ouvrir le port HTTPS du service Web. Vous pouvez utiliser le service `firewall https` à cette fin.

```
- name: Open the port for the web server
  firewalld:
    service: https
    state: enabled
    immediate: true
    permanent: true
```

10. Ajoutez une huitième tâche utilisant le module `service` pour activer et redémarrer le service Web sur l'hôte géré pour que toutes les modifications de configuration prennent effet. Le nom du service Web est défini par la variable `web_svc`.

Ajoutez les lignes suivantes au playbook pour définir une tâche utilisant le module `service` afin d'activer et redémarrer le service Web.

```
- name: Web service enabled and restarted
  service:
    name: "{{ web_svc }}"
    state: restarted
    enabled: true
```

11. Ajoutez une tâche finale utilisant le module `service` pour activer et redémarrer le service de pare-feu sur l'hôte géré afin que toutes les modifications de configuration prennent effet.

- 11.1. Ajoutez les lignes suivantes au playbook pour définir une tâche utilisant le module `service` afin d'activer et redémarrer le service de pare-feu.

```
- name: Firewall service enable and restarted
  service:
    name: firewalld
    state: restarted
```

```
enabled: true
```

11.2. Le playbook terminé doit se présenter comme suit :

```
---
- name: Install and configure webserver with basic auth
  hosts: webserver
  vars:
    firewall_pkg: firewalld
    web_pkg: httpd
    web_svc: httpd
    ssl_pkg: mod_ssl
    python_pkg: python-passlib
    httpdconf_src: files/httpd.conf
    httpdconf_file: /etc/httpd/conf/httpd.conf
    htaccess_src: files/.htaccess
    secrets_dir: /etc/httpd/secrets
    secrets_file: "{{ secrets_dir }}/htpasswd"
    web_root: /var/www/html
    web_user: guest
  vars_files:
    - vars/secret.yml
  tasks:
    - name: Latest version of necessary packages installed
      yum:
        name:
          - "{{ firewall_pkg }}"
          - "{{ web_pkg }}"
          - "{{ ssl_pkg }}"
          - "{{ python_pkg }}"
        state: latest
    - name: Configure web service
      copy:
        src: "{{ httpdconf_src }}"
        dest: "{{ httpdconf_file }}"
        owner: root
        group: root
        mode: 0644
    - name: Secrets directory exists
      file:
        path: "{{ secrets_dir }}"
        state: directory
        owner: apache
        group: apache
        mode: 0500
    - name: Web user exists in secrets file
      htpasswd:
        path: "{{ secrets_file }}"
        name: "{{ web_user }}"
        password: "{{ web_pass }}"
        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: Open the port for the web server
  firewalld:
    service: https
    state: enabled
    immediate: true
    permanent: true
- name: Web service enabled and restarted
  service:
    name: "{{ web_svc }}"
    state: restarted
    enabled: true
- name: Firewall service enable and restarted
  service:
    name: firewalld
    state: restarted
    enabled: true
```

- 11.3. Quittez l'éditeur de texte et enregistrez le playbook **playbook.yml**.
12. Créez un fichier chiffré avec Ansible Vault, nommé **vars/secret.yml**. Il convient de définir la variable **web\_pass** sur **redhat**, qui sera le mot de passe de l'utilisateur Web.
- 12.1. Créez un sous-répertoire nommé **vars** dans le répertoire de projet.

```
[student@workstation data-review]$ mkdir vars
```

- 12.2. Créez le fichier de variable chiffré, **vars/secret.yml**, en utilisant Ansible Vault. Définissez le mot de passe du fichier chiffré sur **redhat**.

```
[student@workstation data-review]$ ansible-vault create vars/secret.yml
New Vault password: redhat
Confirm New Vault password: redhat
```

- 12.3. Ajoutez la définition de variable suivante au fichier.
- ```
web_pass: redhat
```
- 12.4. Quittez l'éditeur de fichier et enregistrez les modifications dans le fichier.
13. Exécutez le playbook **playbook.yml**.
- 13.1. Avant d'exécuter le playbook, vérifiez que sa syntaxe est correcte à l'aide de la commande **ansible-playbook --syntax-check**. Utilisez **--ask-vault-pass**

pour être invité à entrer le mot de passe du coffre-fort. Saisissez **redhat** lorsque le mot de passe vous est demandé. En cas d'erreurs, corrigez-les avant de passer à l'étape suivante. Le résultat doit ressembler à ce qui suit :

```
[student@workstation data-review]$ ansible-playbook --syntax-check \
> --ask-vault-pass playbook.yml
Vault password: redhat

playbook: playbook.yml
```

- 13.2. À l'aide de la commande **ansible-playbook**, exécutez le playbook avec l'option **--ask-vault-pass**. Saisissez **redhat** lorsque le mot de passe vous est demandé.

```
[student@workstation data-review]$ ansible-playbook playbook.yml --ask-vault-pass
Vault password: redhat
PLAY [Install and configure webserver with basic auth] ****
...output omitted...

PLAY RECAP ****
serverb.lab.example.com      : ok=10    changed=9     unreachable=0    failed=0
```

14. Sur **workstation**, utilisez **curl** pour vérifier que le serveur Web est joignable via HTTPS. Vérifiez que l'authentification de base fonctionne en vous authentifiant en tant qu'utilisateur **guest** et en utilisant **redhat** comme mot de passe.

- 14.1. Sur **workstation**, utilisez **curl** pour vérifier que le serveur Web a été démarré et qu'il est joignable. Utilisez l'option **-u** pour spécifier **guest** en tant qu'utilisateur pour l'authentification. Utilisez également l'option **-k** pour désactiver la vérification du certificat SSL du serveur Web. Lorsqu'un message vous y invite, saisissez **redhat** comme mot de passe.

Si le message suivant apparaît, cela signifie que le serveur Web a été installé, que le pare-feu a été mis à jour à l'aide d'une nouvelle règle et que l'authentification de base fonctionne.

```
[student@workstation data-review]$ curl https://serverb.lab.example.com \
> -k -u guest
Enter host password for user 'guest': redhat
serverb.lab.example.com (172.25.250.11) has been customized by Ansible
```

## Évaluation

Sur **workstation**, exécutez la commande **lab data-review grade** pour confirmer que l'exercice est réussi. Corrigez toute erreur signalée et répétez le script tant que des erreurs persistent.

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

## Nettoyage

À partir de **workstation**, exécutez le script **lab data-review cleanup** pour effacer l'exercice.

```
[student@workstation ~]$ lab data-review cleanup
```

L'atelier est maintenant terminé.

# RÉSUMÉ

---

Dans ce chapitre, vous avez appris les principes suivants :

- Les *variables* Ansible permettent aux administrateurs de réutiliser des valeurs dans d'autres fichiers d'un projet Ansible.
- Des variables peuvent être définies pour les hôtes et les groupes d'hôtes dans le fichier d'inventaire.
- Les variables peuvent être définies pour les playbooks en utilisant des faits et des fichiers externes. Elles peuvent également être définies sur la ligne de commande.
- Le mot-clé **register** permet de capturer la sortie d'une commande dans une variable.
- Il est préférable de stocker les variables d'inventaire dans des fichiers dans le répertoire **host\_vars** et **group\_vars** correspondant à l'inventaire, plutôt que dans le fichier d'inventaire lui-même.
- Ansible Vault constitue un moyen de protéger des données sensibles comme les hachages de mot de passe et les clés privées pour le déploiement à l'aide de playbooks Ansible.
- Ansible Vault peut être utilisé pour créer et chiffrer un fichier texte s'il n'existe pas déjà ou pour chiffrer et déchiffrer des fichiers existants.
- Red Hat recommande aux utilisateurs de conserver la majeure partie des variables dans un fichier normal et les variables sensibles dans un second fichier protégé par Ansible Vault.
- Les faits *Ansible* sont des variables détectées automatiquement par Ansible sur un hôte géré.

## CHAPITRE 5

# MISE EN ŒUVRE D'UN CONTRÔLE DE TÂCHE

### PROJET

Gérer le contrôle de tâche, les gestionnaires et les erreurs de tâche dans les playbooks Ansible.

### OBJECTIFS

- Utiliser des boucles pour écrire des tâches efficaces et des conditions pour contrôler quand exécuter des tâches.
- Implémenter une tâche qui s'exécute uniquement lorsqu'une autre tâche modifie l'hôte géré.
- Contrôler ce qui se passe lorsqu'une tâche échoue et quelles conditions provoquent son échec.

### SECTIONS

- Écriture de boucles et de tâches conditionnelles (et exercice guidé)
- Mise en œuvre de gestionnaires (et exercice guidé)
- Gestion des échecs de tâche (et exercice guidé)

### ATELIER

- Mise en œuvre d'un contrôle de tâche

# ÉCRITURE DE BOUCLES ET DE TÂCHES CONDITIONNELLES

---

## OBJECTIFS

Au terme de cette section, les stagiaires doivent pouvoir utiliser des boucles pour écrire des tâches efficaces et des conditions pour contrôler quand exécuter des tâches.

## ITÉRATION DE TÂCHES À L'AIDE DE BOUCLES

L'utilisation de boucles évite aux administrateurs de devoir écrire plusieurs tâches utilisant le même module. Par exemple, au lieu d'écrire cinq tâches pour s'assurer de l'existence de cinq utilisateurs, vous pouvez écrire une tâche itérant une liste de cinq utilisateurs.

Ansible prend en charge l'itération d'une tâche sur un ensemble d'éléments à l'aide du mot-clé **loop**. Vous pouvez configurer des boucles pour répéter une tâche à l'aide de chaque élément d'une liste, du contenu de chaque fichier d'une liste, d'une séquence générée de nombres ou de structures plus compliquées. Cette section couvre les boucles simples effectuant une itération sur une liste d'éléments. Consultez la documentation pour des scénarios de bouclage plus avancés.

### Boucles simples

Une boucle simple itère une tâche sur une liste d'éléments. Le mot-clé **loop** est ajouté à la tâche et prend comme valeur la liste des éléments que la tâche doit itérer. La variable de boucle **item** convient la valeur utilisée lors de chaque itération.

Considérez l'extrait de code suivant qui utilise à deux reprises le module `service` afin de s'assurer que les deux services réseau sont en cours d'exécution :

```
- name: Postfix is running
  service:
    name: postfix
    state: started

- name: Dovecot is running
  service:
    name: dovecot
    state: started
```

Vous pouvez réécrire ces deux tâches avec une boucle simple de sorte qu'une seule tâche suffise à s'assurer que les deux services sont en cours d'exécution :

```
- name: Postfix and Dovecot are running
  service:
    name: "{{ item }}"
    state: started
  loop:
    - postfix
    - dovecot
```

La liste utilisée par **loop** peut être fournie par une variable. Dans l'exemple suivant, la variable **mail\_services** contient la liste des services qui doivent être exécutés.

```
vars:  
  mail_services:  
    - postfix  
    - dovecot  
  
tasks:  
  - name: Postfix and Dovecot are running  
    service:  
      name: "{{ item }}"  
      state: started  
    loop: "{{ mail_services }}"
```

## Boucles sur une liste de hachages/dictionnaires

La liste **loop** ne doit pas forcément être une liste de valeurs simples. Dans l'exemple suivant, chaque élément de la liste est en fait un hachage/dictionnaire. Chaque hachage/dictionnaire de l'exemple contient deux clés, **name** et **groups**, et la valeur de chaque clé dans la variable de boucle **item** peut être récupérée avec les variables **item.name** et **item.groups**, respectivement.

```
- 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
```

Le résultat de la tâche précédente affiche la présence de l'utilisatrice **jane**, membre du groupe **wheel**, ainsi que la présence de l'utilisateur **joe**, membre du groupe **root**.

## Mots-clés de boucle de style antérieur

Avant Ansible 2.5, la plupart des playbooks utilisaient une syntaxe différente pour les boucles. Plusieurs mots-clés de boucle ont été fournis, avec le préfixe **with\_**, suivi du nom d'un plug-in de recherche Ansible (une fonctionnalité avancée qui n'est pas couverte en détail dans ce cours). Cette syntaxe de bouclage est très courante dans les playbooks existants, mais sera probablement déconseillée à un moment donné dans le futur.

Quelques exemples sont énumérés dans le tableau ci-dessous :

## Boucles Ansible de style antérieur

| MOT-CLÉ LOOP         | DESCRIPTION                                                                                                                                                                                                                                                                                                                                                                                          |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>with_items</b>    | Se comporte de la même façon que le mot-clé <b>loop</b> pour des listes simples, telles qu'une liste de chaînes ou une liste de hachages/dictionnaires. Contrairement à <b>loop</b> , si des listes de listes sont fournies à <b>with_items</b> , elles sont aplatis dans une liste à niveau unique. La variable de boucle <b>item</b> convient l'élément de liste utilisé lors de chaque itération. |
| <b>with_file</b>     | Ce mot-clé nécessite une liste de noms de fichiers de nœuds de contrôle. La variable de boucle <b>item</b> contient le contenu d'un fichier correspondant de la liste de fichiers à chaque itération.                                                                                                                                                                                                |
| <b>with_sequence</b> | Au lieu de requérir une liste, ce mot-clé nécessite des paramètres pour générer une liste de valeurs basée sur une séquence numérique. La variable de boucle <b>item</b> contient la valeur de l'un des éléments générés dans la séquence générée à chaque itération.                                                                                                                                |

Ci-dessous un exemple de **with\_items** dans un playbook :

```
vars:
  data:
    - user0
    - user1
    - user2
tasks:
  - name: "with_items"
    debug:
      msg: "{{ item }}"
    with_items: "{{ data }}"
```



### IMPORTANT

Depuis Ansible 2.5, la méthode recommandée pour écrire des boucles consiste à utiliser le mot-clé **loop**.

Cependant, vous devez toujours comprendre l'ancienne syntaxe, en particulier **with\_items**, car elle est largement utilisée dans les playbooks existants. Vous êtes susceptible de rencontrer des playbooks et des rôles qui continuent à utiliser des mots-clés **with\_\*** pour le bouclage.

Toute tâche utilisant l'ancienne syntaxe peut être convertie pour utiliser **loop** en conjonction avec les filtres Ansible. Pour ce faire, vous n'avez pas besoin de savoir comment utiliser les filtres Ansible. Il existe une bonne référence sur la façon de convertir les anciennes boucles en nouvelle syntaxe, ainsi que des exemples sur la manière de boucler des éléments qui ne sont pas de simples listes, dans la documentation Ansible dans la section « Migration à partir de with\_X vers la boucle » [[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks\\_loops.html#migrating-from-with-x-to-loop](https://docs.ansible.com/ansible/2.7/user_guide/playbooks_loops.html#migrating-from-with-x-to-loop)] du *Guide de l'utilisateur Ansible*.

Vous rencontrerez probablement des tâches provenant de playbooks plus anciens contenant des mots-clés **with\_\***.

Les techniques avancées de bouclage dépassent le cadre de ce cours. Toutes les tâches d'itération de ce cours peuvent être implémentées avec le mot-clé **with\_items** ou **loop**.

## Utilisation de variables de registre avec des boucles

Le mot-clé **register** peut également capturer le résultat d'une tâche en boucle. L'extrait de code suivant montre la structure de la variable de registre à partir d'une tâche en boucle :

```
[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** La variable **echo\_results** est enregistrée.
- ②** Le contenu de la variable **echo\_results** est affiché à l'écran.

L'exécution du playbook ci-dessus génère le résultat suivant :

```
[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"
        ]
    }
}
]❷
}
...output omitted...

```

- ❶ Le caractère { indique que le début de la variable **echo\_results** est composé de paires clé-valeur.
- ❷ La clé **results** contient les résultats de la tâche précédente. Le caractère [ indique le début d'une liste.
- ❸ Le début des métadonnées de tâche pour le premier élément (indiqué par la clé **item**). La sortie de la commande **echo** se trouve dans la clé **stdout**.
- ❹ Le début des métadonnées de résultat de tâche pour le deuxième élément.
- ❺ Le caractère ] indique la fin de la liste **results**.

Dans ce qui précède, la clé **results** contient une liste. Ci-dessous, le playbook est modifié de sorte que la deuxième tâche itère cette liste :

```
[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']

```

Après avoir exécuté le playbook ci-dessus, le résultat est :

```

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...

```

## EXÉCUTION DE TÂCHES DE MANIÈRE CONDITIONNELLE

Ansible peut utiliser des *conditions* pour exécuter des tâches ou des plays lorsque certaines conditions sont remplies. Par exemple, il est possible d'utiliser une condition pour déterminer la quantité de mémoire disponible sur un hôte géré avant qu'Ansible installe ou configure un service.

Les conditions permettent aux administrateurs de distinguer les hôtes gérés et de leur attribuer des rôles fonctionnels selon des conditions qu'ils remplissent. Les variables de playbook, les variables enregistrées et les faits Ansible peuvent tous être testés au moyen de conditions. Des opérateurs permettant de comparer des chaînes, des données numériques et des valeurs booléennes sont disponibles.

Les scénarios suivants illustrent l'utilisation de conditions dans Ansible :

- Une limite fixe peut être définie dans une variable (par exemple, `min_memory`) afin de la comparer à la mémoire disponible sur un hôte géré.
- La sortie d'une commande peut être capturée et évaluée par Ansible afin de déterminer si une tâche est terminée avant de démarrer une nouvelle action. Par exemple, si un programme échoue, un lot est ignoré.
- Utilisez les faits Ansible pour déterminer la configuration réseau d'un hôte géré et décider quel fichier de modèle envoyer (par exemple, association ou troncation de réseau).
- Le nombre de processeurs peut être évalué pour déterminer comment régler correctement un serveur Web.

- Comparez une variable enregistrée avec une variable prédéfinie pour déterminer si un service a changé. Par exemple, testez la somme de contrôle MD5 d'un fichier de configuration de service pour voir si le service a changé.

## Syntaxe de tâche conditionnelle

L'instruction **when** sert à exécuter une tâche de manière conditionnelle. Il prend comme valeur la condition à tester. Si la condition est respectée, la tâche est exécutée. À l'inverse, si la condition n'est pas respectée, la tâche est ignorée.

L'une des conditions les plus simples pouvant être testée est l'état « true » ou « false » d'une variable booléenne. L'instruction **when** de l'exemple suivant entraîne l'exécution de la tâche uniquement si **run\_my\_task** a la valeur « true » :

```
---
- 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
```

L'exemple suivant est un peu plus compliqué, car il teste si la variable **my\_service** contient une valeur. Le cas échéant, la valeur de **my\_service** est utilisée comme nom du paquetage à installer. Si la variable **my\_service** n'est pas définie, la tâche est alors ignorée sans erreur.

```
---
- 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
```

Dans le tableau suivant figurent quelques-unes des opérations que les administrateurs peuvent appliquer lorsqu'ils utilisent des conditions :

### Exemples de conditions

| OPÉRATION                       | EXAMPLE                            |
|---------------------------------|------------------------------------|
| Égal (la valeur est une chaîne) | <b>ansible_machine == "x86_64"</b> |
| Égal (valeur numérique)         | <b>max_memory == 512</b>           |

| OPÉRATION                                                                                                                                                                                                                                                                    | EXEMPLE                                                  |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------|
| Inférieur à                                                                                                                                                                                                                                                                  | <code>min_memory &lt; 128</code>                         |
| Supérieur à                                                                                                                                                                                                                                                                  | <code>min_memory &gt; 256</code>                         |
| Inférieur ou égal à                                                                                                                                                                                                                                                          | <code>min_memory &lt;= 256</code>                        |
| Supérieur ou égal à                                                                                                                                                                                                                                                          | <code>min_memory &gt;= 512</code>                        |
| Different de                                                                                                                                                                                                                                                                 | <code>min_memory != 512</code>                           |
| La variable existe                                                                                                                                                                                                                                                           | <code>min_memory is defined</code>                       |
| La variable n'existe pas                                                                                                                                                                                                                                                     | <code>min_memory n'est pas défini</code>                 |
| La variable booléenne est égale à <code>true</code> . Les valeurs de <code>1</code> , <code>True</code> ou <code>yes</code> sont évaluées sur <code>true</code> . Les valeurs de <code>0</code> , <code>False</code> ou <code>no</code> sont évaluées à <code>false</code> . | <code>memory_available</code>                            |
| La variable booléenne est égale à <code>false</code> .                                                                                                                                                                                                                       | <code>not memory_available</code>                        |
| La valeur de la première variable est présente en tant que valeur de la liste de la seconde variable                                                                                                                                                                         | <code>ansible_distribution dans supported_distros</code> |

La dernière entrée du tableau précédent peut être troublante au premier abord. L'exemple suivant illustre son fonctionnement.

Dans l'exemple, la variable `ansible_distribution` est un fait déterminé au cours de la tâche **Gathering Facts** (Collecte des faits) et identifie la distribution du système d'exploitation de l'hôte géré. La variable `supported_distros` a été créée par l'auteur du playbook et contient une liste des distributions de système d'exploitation prises en charge par ce dernier. Si la valeur de `ansible_distribution` figure dans la liste `supported_distros`, la condition passe et la tâche est exécutée.

```

---
- name: Demonstrate the "in" keyword
  hosts: all
  vars:
    supported_distros:
      - RedHat
      - Fedora
  tasks:
    - name: Install httpd using yum, where supported
      yum:
        name: http
        state: present
      when: ansible_distribution in supported_distros

```

**IMPORTANT**

Notez la mise en retrait de l'instruction **when**. L'instruction **when** n'étant pas une variable de module, elle doit être placée hors du module, en retrait, au niveau supérieur de la tâche.

Une tâche est un hachage/dictionnaire YAML et l'instruction **when** n'est qu'une clé supplémentaire dans la tâche telle le nom de celle-ci et le module qu'elle utilise. Une convention d'usage place le mot-clé **when** qui peut être présente après le nom de la tâche et le module (et les arguments du module).

## Conditions multiples

Vous pouvez utiliser une instruction **when** pour évaluer plusieurs conditions. Pour ce faire, des conditions peuvent être combinées soit avec le mot-clé **and** soit avec le mot-clé **or**, et groupés à l'aide de parenthèses.

Les snippets suivants montrent comment exprimer plusieurs conditions.

- Si, pour que soit rempli un argument conditionnel, il suffit qu'une seule condition soit vraie, vous devez utiliser l'argument **or**. Par exemple, la condition suivante est remplie si l'ordinateur exécute soit Red Hat Enterprise Linux soit Fedora :

```
when: ansible_distribution == "RedHat" or ansible_distribution == "Fedora"
```

- Avec l'opération **and**, les deux conditions doivent être vraies pour que l'ensemble de l'argument conditionnel soit rempli. Par exemple, la condition suivante sera remplie si l'hôte distant est un hôte Red Hat Enterprise Linux 7.5, et le noyau installé est la version spécifiée :

```
when: ansible_distribution_version == "7.5" and ansible_kernel == "3.10.0-327.el7.x86_64"
```

Le mot-clé **when** prend également en charge l'utilisation d'une liste pour décrire une liste de conditions. Lorsqu'une liste est fournie au mot-clé **when**, toutes les conditions sont combinées en utilisant l'opération **and**. L'exemple ci-dessous illustre une autre façon de combiner plusieurs instructions conditionnelles à l'aide de l'opérateur **and** :

```
when:
  - ansible_distribution_version == "7.5"
  - ansible_kernel == "3.10.0-327.el7.x86_64"
```

Ce format améliore la lisibilité, un objectif clé de playbooks Ansible bien écrits.

- Des instructions conditionnelles plus complexes peuvent être exprimées en regroupant des conditions avec des parenthèses. Cela garantit qu'elles soient correctement interprétées.

Par exemple, la condition suivante est remplie si l'ordinateur exécute soit Red Hat Enterprise Linux 7 soit Fedora 28 : Cet exemple utilise le caractère plus grand que (>) de sorte que la longue condition puisse être divisée sur plusieurs lignes dans le playbook, afin de faciliter sa lecture.

```
when: >
  ( ansible_distribution == "RedHat" and
```

```
ansible_distribution_major_version == "7" )
or
( ansible_distribution == "Fedora" and
ansible_distribution_major_version == "28" )
```

## COMBINAISON DE BOUCLES ET DE TÂCHES CONDITIONNELLES

Vous pouvez associer des boucles et des conditions.

Dans l'exemple suivant, le paquetage *mariadb-server* est installé par le module **yum** si un système de fichiers est monté sur / avec plus de 300 Mo d'espace libre. Le fait **ansible\_mounts** est une liste de dictionnaires, représentant chacun des faits sur un système de fichiers monté. La boucle répète chaque dictionnaire de la liste, et l'argument conditionnel n'est pas rempli à moins qu'un dictionnaire représente un système de fichiers monté au sein duquel les deux conditions sont vraies.

```
- 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
```



### IMPORTANT

Quand vous utilisez **when** avec **loop** pour une tâche, l'instruction **when** est vérifiée pour chaque élément.

Voici un autre exemple dans lequel sont combinées des conditions et des variables sont enregistrées. Le playbook annoté suivant redémarrera le service *httpd* uniquement si le service *postfix* est exécuté.

```
---
- name: Restart HTTPD if Postfix is Running
  hosts: all
  tasks:
    - name: Get Postfix server status
      command: /usr/bin/systemctl is-active postfix ❶
      ignore_errors: yes❷
      register: result❸
    - name: Restart Apache HTTPD based on Postfix status
      service:
        name: httpd
        state: restarted
      when: result.rc == 0❹
```

**❶** Le service Postfix est-il exécuté ?

**❷** S'il n'est pas exécuté et que la commande échoue, n'arrêtez pas le traitement

- ③ Enregistre des informations sur le résultat du module dans une variable nommée `result`.
- ④ Évalue le résultat de la tâche Postfix. Si le code de sortie de la commande `systemctl` est 0, alors le service Postfix est actif et cette tâche redémarre le service `httpd`.



## RÉFÉRENCES

### Boucles – Documentation Ansible

[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks\\_loops.html](https://docs.ansible.com/ansible/2.7/user_guide/playbooks_loops.html)

### Tests – Documentation Ansible

[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks\\_tests.html](https://docs.ansible.com/ansible/2.7/user_guide/playbooks_tests.html)

### Conditions – Documentation Ansible

[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks\\_conditionals.html](https://docs.ansible.com/ansible/2.7/user_guide/playbooks_conditionals.html)

### Validité des noms de variables – Variables – Documentation Ansible

[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks\\_variables.html#what-makes-a-valid-variable-name](https://docs.ansible.com/ansible/2.7/user_guide/playbooks_variables.html#what-makes-a-valid-variable-name)

## ► EXERCICE GUIDÉ

# ÉCRITURE DE BOUCLES ET DE TÂCHES CONDITIONNELLES

Dans cet exercice, vous allez écrire un playbook contenant des tâches comportant des conditions et des boucles.

## RÉSULTATS

Vous devez pouvoir réaliser les tâches suivantes :

- Mettre en œuvre des conditions Ansible à l'aide du mot-clé **when**.
- Implémenter l'itération de tâches à l'aide du mot-clé **loop** en conjonction avec les conditions.

## Présentation du scénario

Cet exercice contient un playbook, **database\_setup.yml**, qui installe une base de données et crée des comptes d'utilisateur sur des hôtes distants désignés. Tel que mis en œuvre, le playbook ne peut prendre en charge que l'installation d'une base de données sur un hôte Red Hat Enterprise Linux.

Grâce à l'utilisation de tâches conditionnelles, le playbook est structuré de manière à permettre l'ajout progressif d'autres distributions Linux ultérieurement. Les tâches associées à la création d'utilisateurs sont conservées dans un fichier séparé, **database\_users\_tasks.yml**, car elles peuvent également être utilisées avec d'autres distributions Linux.

Les deux fichiers **database\_setup.yml** et **database\_users\_tasks.yml** sont incomplets. En conséquence, le playbook **database\_setup.yml** n'est pas fonctionnel. Dans cet exercice, vous allez implémenter les mots-clés **loop** et **when** pour les tâches de ces fichiers afin de créer un playbook fonctionnant correctement.

Sur **workstation**, exécutez le script de configuration de l'atelier pour vérifier que l'environnement est prêt pour commencer l'atelier. Le script crée le répertoire de travail **control-flow** et le renseigne à l'aide d'un fichier de configuration Ansible, d'un inventaire d'hôtes et de fichiers de playbook partiellement terminés.

```
[student@workstation ~]$ lab control-flow setup
```

- 1. En tant qu'utilisateur **student** sur **workstation**, accédez au répertoire **/home/student/control-flow**.

```
[student@workstation ~]$ cd ~/control-flow
[student@workstation control-flow]$
```

- 2. Examinez le fichier du playbook **database\_setup.yml**. Ajoutez les instructions conditionnelles appropriées à chaque tâche du fichier de playbook.

2.1. Utilisez un éditeur de texte pour passer en revue le fichier **database\_setup.yml**.

```

---
- name: Database Setup play
hosts: database_servers
vars:
  min_ram_size_bytes: 20000000000 1
  supported_distros: 2
    - RedHat
    #- Centos
tasks:
  - name: Setup Database tasks on supported hosts w/ Min. RAM
    include_tasks: "{{ ansible_distribution }}_database_tasks.yml"
    #Add a conditional here 3

  - name: Print a message for unsupported Distros
    debug:
      msg: >
        {{ inventory_hostname }} is a
        {{ ansible_distribution }}-based host, which is not one
        of the supported distributions ({{ supported_distros }})

    #Add a conditional here 4

  - name: Print a message for systems with insufficient RAM
    debug:
      msg: >
        {{ inventory_hostname }} does not meet the minimum
        RAM requirements of {{ min_ram_size_bytes }} bytes.

  #Add a conditional here 5

```

**1** **2** Ces variables définissent la configuration RAM minimale requise pour les serveurs de base de données et une liste des distributions Linux prises en charge pour les hôtes de base de données. Vos serveurs de base de données doivent avoir au moins 2 Go de RAM et la liste des distributions prises en charge est actuellement un seul élément : **RedHat**(correspondant à Red Hat Enterprise Linux).

**3** La première tâche, qui comprend les tâches d'installation d'une base de données, ne doit être exécutée que si le serveur distant répond à deux critères. Tout d'abord, le système d'exploitation du serveur distant doit être l'une des distributions spécifiées par la variable **supported\_distros**. Deuxièmement, le serveur distant doit disposer de la RAM minimale spécifiée par la variable **min\_ram\_size\_bytes** du playbook. Si l'un ou l'autre de ces critères n'est pas

rempli, les tâches d'installation de la base de données ne sont pas incluses dans le play.

- ④ La deuxième tâche imprime un message si un serveur distant ne correspond pas à l'une des distributions de système d'exploitation prises en charge.
- ⑤ La troisième tâche imprime un message si un serveur distant ne répond pas aux exigences en matière de RAM pour un serveur de base de données, spécifié par la variable `min_ram_size_bytes`.

## 2.2. Modifiez la première tâche du fichier `database_setup.yml`.

```
- name: Setup Database tasks on supported hosts w/ Min. RAM
  include_tasks: "{{ ansible_distribution }}_database_tasks.yml"
  #Add a conditional here
```

Remplacez le commentaire **#Add a conditional here** par une instruction **when** pour tester la distribution Linux et les exigences en matière de RAM. La variable `ansible_memtotal_mb` est un fait Ansible qui fournit la mémoire vive de l'hôte distant, en mébioc tet (MiB, mégaoctets binaires). Rappelons que 1 MiB est égal à  $1024 * 1024$  (1 048 576) octets.

Après l'avoir modifié, enregistrez le fichier. La tâche doit maintenant se présenter comme suit :

```
- name: Setup Database tasks on supported hosts w/ Min. RAM
  include_tasks: "{{ ansible_distribution }}_database_tasks.yml"
  when:
    - ansible_distribution in supported_distros
    - ansible_memtotal_mb*1024*1024 >= min_ram_size_bytes
```

## 2.3. Modifiez la seconde tâche du fichier `database_setup.yml`.

```
- name: Print a message for unsupported Distros
  debug:
    msg: >
      {{ inventory_hostname }} is a
      {{ ansible_distribution }}-based host, which is not one
      of the supported distributions ({{ supported_distros }})
  #Add a conditional here
```

Remplacez **#Add a conditional here** par une instruction **when** pour vérifier que l'hôte distant ne fait pas partie des distributions prises en charge. Après l'avoir modifié, enregistrez le fichier. La tâche doit maintenant se présenter comme suit :

```
- name: Print a message for unsupported Distros
  debug:
    msg: >
      {{ inventory_hostname }} is a
      {{ ansible_distribution }}-based host, which is not one
      of the supported distributions ({{ supported_distros }})
```

```
when: ansible_distribution not in supported_distros
```

## 2.4. Modifiez la troisième tâche du fichier **database\_setup.yml**.

```
- name: Print a message for systems with insufficient RAM
  debug:
    msg: >
      {{ inventory_hostname }} does not meet the minimum
      RAM requirements of {{ min_ram_size_bytes }} bytes.
  #Add a conditional here
```

Remplacez le commentaire **#Add a conditional here** par une instruction **when** pour vérifier si l'hôte distant répond aux exigences en matière de RAM. Après l'avoir modifié, enregistrez le fichier. La tâche doit maintenant se présenter comme suit :

```
- name: Print a message for systems with insufficient RAM
  debug:
    msg: >
      {{ inventory_hostname }} does not meet the minimum
      RAM requirements of {{ min_ram_size_bytes }} bytes.
  when: ansible_memtotal_mb*1024*1024 < min_ram_size_bytes
```

Lorsque vous avez terminé les modifications, le fichier **database\_setup.yml** doit contenir les éléments suivants :

```
---
- name: Database Setup play
  hosts: database_servers
  vars:
    min_ram_size_bytes: 2000000000
    supported_distros:
      - RedHat
      #- Centos
  tasks:
    - name: Setup Database tasks on supported hosts w/ Min. RAM
      include_tasks: "{{ ansible_distribution }}_database_tasks.yml"
      when:
        - ansible_distribution in supported_distros
        - ansible_memtotal_mb*1024*1024 >= min_ram_size_bytes

    - name: Print a message for unsupported Distros
      debug:
        msg: >
          {{ inventory_hostname }} is a
          {{ ansible_distribution }}-based host, which is not one
          of the supported distributions ({{ supported_distros }})
      when: ansible_distribution not in supported_distros

    - name: Print a message for systems with insufficient RAM
      debug:
        msg: >
          {{ inventory_hostname }} does not meet the minimum
          RAM requirements of {{ min_ram_size_bytes }} bytes.
```

```
when: ansible_memtotal_mb*1024*1024 < min_ram_size_bytes
```

- 3. Si l'hôte distant répond aux exigences en matière de RAM et est installé avec RedHat Enterprise Linux, les tâches du fichier **RedHat\_database\_tasks.yml** sont utilisées pour effectuer l'installation de la base de données. Examinez le fichier **RedHat\_database\_tasks.yml**.

```
#For RHEL, using mariadb as the database service
- name: Set the 'db_service' fact
  set_fact:①
    db_service: mariadb

#RHEL packages for mariadb service
- name: Ensure database packages are installed
  yum:
    name:②
      - mariadb-server
      - mariadb-bench
      - mariadb-libs
      - mariadb-test

- name: Ensure the database service is started
  service:
    name: "{{ db_service }}"
    state: started
    enabled: true

#Below tasks are also reused by other distros
- name: Create Database Users
  include_tasks: database_user_tasks.yml③
```

- ① Le module `set_fact` définit une variable, `db_service`, définie sur **mariadb**. Cela permet à chaque distribution d'utiliser un service de base de données différent.
  - ② Une simple boucle ne doit pas être utilisée pour installer plusieurs paquetages. Certains modules, tels que le module `yum`, prennent en charge l'utilisation d'une liste comme valeur de paramètre.
  - ③ Une liste de tâches est incluse pour la création de comptes d'utilisateurs sur l'hôte de la base de données distante. Ces tâches sont enregistrées dans un fichier séparé afin de faciliter leur réutilisation pour d'autres distributions Linux.
- 4. Modifiez le fichier de tâche **database\_users\_tasks.yml**. Mettez à jour la première tâche avec une boucle simple pour créer tous les groupes de permissions définis dans la variable de groupe `host_permission_groups`. La variable `host_permission_groups` est définie dans le fichier **group\_vars/database\_servers.yml**.

#### 4.1. Parcourez le contenu du fichier **group\_vars/database\_servers.yml**.

```
[student@workstation control-flow]$ cat group_vars/database_servers.yml
host_permission_groups:
  - dbadmin
```

```
- dbuser
```

Le fichier définit une variable de liste simple, `host_permission_groups`, qui contient les noms de groupe. Ces noms de groupe s'appliquent uniquement aux hôtes du groupe d'hôtes **database\_servers**.

- 4.2. Ouvrez le fichier de tâche **database\_users\_tasks.yml** dans un éditeur. Modifiez la première tâche dans le fichier de tâche **database\_users\_tasks.yml**.

```
- name: Ensure database permission groups exist
group:
  name:
  state: present
#Add a loop
```

- 4.3. Remplacez la ligne **#Add a loop** par une instruction loop pour itérer les valeurs des `host_permission_groups`. Remplacez la valeur du mot-clé **name** par la variable d'itération de boucle appropriée.

La première tâche contient maintenant :

```
- name: Ensure database permission groups exist
group:
  name: "{{ item }}"
  state: present
loop: "{{ host_permission_groups }}"
```

- 5. Mettez à jour la deuxième tâche du fichier **database\_users\_tasks.yml** pour effectuer une boucle sur les utilisateurs dans la variable `user_list`. La variable `user_list` est définie pour tous les hôtes d'inventaire du fichier **group\_vars/all.yml**.

Ajoutez chaque utilisateur créé à un groupe identique à la valeur **role** de l'utilisateur. Utilisez la valeur **username** de l'utilisateur pour la valeur du mot-clé **name** dans le module `user`.

Enfin, ajoutez une instruction **when** pour vous assurer qu'un utilisateur est créé uniquement si son rôle correspond à l'une des valeurs présentes dans la variable `host_permission_groups`.

- 5.1. Parcourez le contenu du fichier **group\_vars/all.yml**.

```
[student@workstation control-flow]$ cat group_vars/all.yml
user_list:
  - name: John Davis
    username: jdavis
    role: dbadmin
  - name: Jennifer Smith
    username: jsmith
    role: dbuser
...output omitted...
```

Le fichier définit une variable de liste, `user_list`, qui contient un hachage/dictionnaire représentant un utilisateur unique. Pour chaque utilisateur, les attributs **name**, **username** et **role** sont définis. Dans cet exercice, l'attribut **role** correspond

**CHAPITRE 5** | Mise en œuvre d'un contrôle de tâche

à un groupe de permissions Linux. Cette variable est définie pour tous les hôtes de l'inventaire, y compris les hôtes du groupe d'hôtes **database\_servers**.

- 5.2. Examinez la seconde tâche du fichier **database\_users\_tasks.yml**.

```
- name: Ensure Database Users exist
  user:
    name:
    groups:
    append: yes
    state: present
```

- 5.3. Ajoutez une instruction **loop** pour itérer les valeurs dans la variable **user\_list**.

```
loop: "{{ user_list }}"
```

- 5.4. Mettez à jour la valeur du mot-clé **name** du module **user** pour qu'elle corresponde au **username** de l'utilisateur.

```
name: "{{ item.username }}"
```

- 5.5. Mettez à jour la valeur du mot-clé **groups** du module **user** pour qu'elle corresponde au **role** de l'utilisateur.

```
groups: "{{ item.role }}"
```

- 5.6. Ajoutez une instruction **when** à la deuxième tâche qui vérifie si le **role** d'un utilisateur correspond à l'une des valeurs de la variable **host\_permission\_groups**.

```
when: item.role in host_permission_groups
```

La deuxième tâche devrait maintenant contenir les éléments suivants :

```
- name: Ensure Database Users exist
  user:
    name: "{{ item.username }}"
    groups: "{{ item.role }}"
    append: yes
    state: present
  loop: "{{ user_list }}"
  when: item.role in host_permission_groups
```

- 6. Vérifiez la syntaxe du playbook **database\_setup.yml** à l'aide de la commande **ansible-playbook --syntax-check**:

```
[student@workstation control-flow]$ ansible-playbook \
> --syntax-check database_setup.yml
playbook: database_setup.yml
```

- 7. Exécutez le playbook **database\_setup.yml** à l'aide de la commande **ansible-playbook**.

```
[student@workstation control-flow]$ ansible-playbook database_setup.yml

PLAY [Database Setup play] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Setup Database tasks on supported hosts w/ Min. RAM] ****
skipping: [servera.lab.example.com]

TASK [Print a message for unsupported Distros] ****
skipping: [servera.lab.example.com]

TASK [Print a message for systems with insufficient RAM] ****
ok: [servera.lab.example.com] => {
    "msg": "servera.lab.example.com does not meet the minimum RAM
requirements of 2000000000 bytes.\n"
}

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

La sortie indique que servera ne répond pas aux exigences minimales en matière de RAM. En conséquence, aucune base de données n'est installée sur servera.

- 8. Modifiez les exigences minimales en matière de RAM du playbook de 2 Go à 500 Mo. Exécutez à nouveau le playbook et passez en revue les résultats.

- 8.1. Ouvrez le playbook **database\_setup.yml** dans un éditeur de texte. Mettez à jour la valeur de la variable `min_ram_size_bytes` à **500000000** et enregistrez les modifications. La partie supérieure du playbook correspond maintenant à l'extrait de code ci-dessous :

```
---
- name: Database Setup play
  hosts: database_servers
  vars:
    min_ram_size_bytes: 500000000
    supported_distros:
      - RedHat
```

Enregistrez le fichier.

- 8.2. Exécutez à nouveau le playbook **database\_setup.yml**.

```
[student@workstation control-flow]$ ansible-playbook database_setup.yml

PLAY [Database Setup play] ****
TASK [Gathering Facts] ****
```

```

ok: [servera.lab.example.com]

TASK [Setup Database tasks on supported hosts w/ Min. RAM] ****
included: ...output omitted.../RedHat_database_tasks.yml ...output omitted

TASK [Set the 'db_service' fact] ****
ok: [servera.lab.example.com]

TASK [Ensure database packages are installed] ****
changed: [servera.lab.example.com]

TASK [Ensure the database service is started] ****
changed: [servera.lab.example.com]

TASK [Create Database Users] ****
included: ...output omitted.../database_user_tasks.yml ...output omitted...

TASK [Ensure database permission groups exist] ****
ok: [servera.lab.example.com] => (item=dbadmin)
ok: [servera.lab.example.com] => (item=dbuser)

TASK [Ensure Database Users exist] ****
changed: [servera.lab.example.com] => (item={'username': u'jdavis', 'role': u'dbadmin', 'name': u'John Davis'})
changed: [servera.lab.example.com] => (item={'username': u'jsmith', 'role': u'dbuser', 'name': u'Jennifer Smith'})
skipping: [servera.lab.example.com] => (item={'username': u'sjohnson', 'role': u'webadmin', 'name': u'Sarah Johnson'})
skipping: [servera.lab.example.com] => (item={'username': u'mjones', 'role': u'webdev', 'name': u'Matt Jones'})

TASK [Print a message for unsupported Distros] ****
skipping: [servera.lab.example.com]

TASK [Print a message for systems with insufficient RAM] ****
skipping: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com : ok=8    changed=4    unreachable=0    failed=0

```

La sortie indique qu'une base de données a été installée avec succès. Les utilisateurs avec un rôle lié aux bases de données peuvent maintenant se connecter à servera.

## Nettoyage

Sur workstation, exéutez le script **lab control-flow cleanup** pour effacer les ressources créées au cours de cet exercice.

```
[student@workstation ~]$ lab control-flow cleanup
```

L'exercice guidé est maintenant terminé.

# MISE EN ŒUVRE DES GESTIONNAIRES

---

## OBJECTIFS

Une fois cette section terminée, les stagiaires doivent être en mesure d'implémenter une tâche qui ne s'exécute que lorsqu'une autre tâche modifie l'hôte géré.

**Figure 5.0: Mise en œuvre des gestionnaires**

## GESTIONNAIRES ANSIBLE

Les modules Ansible sont conçus pour être *idempotents*. Cela signifie que dans un playbook bien écrit, le playbook et les tâches correspondantes peuvent être exécutés plusieurs fois sans modifier l'hôte géré, à moins qu'une modification soit nécessaire pour amener l'hôte géré dans un état particulier.

Toutefois, à certains moments lorsqu'une tâche modifie le système, l'exécution d'un tâche supplémentaire peut s'avérer nécessaire. Par exemple, lorsqu'une modification est apportée au fichier de configuration d'un service, il est parfois nécessaire de charger le service une nouvelle fois afin que les modifications prennent effet.

Les *gestionnaires* sont des tâches qui répondent à une notification déclenchée par d'autres tâches. Les tâches notifient leurs gestionnaires uniquement lorsque la tâche modifie quelque chose sur un hôte géré. Chaque gestionnaire possède un nom globalement unique et est déclenché à la fin d'un bloc de tâches dans un playbook. Si aucune tâche ne notifie le gestionnaire par son nom, le gestionnaire ne sera pas exécuté. Si une ou plusieurs tâches notifient le gestionnaire, celui-ci sera exécuté précisément une fois lorsque toutes les autres tâches seront terminées. Les gestionnaires étant des tâches, les administrateurs peuvent utiliser les mêmes modules dans les gestionnaires que dans les autres tâches. En règle générale, les gestionnaires servent à redémarrer les hôtes et à relancer les services.

Les gestionnaires peuvent être considérés comme des tâches *inactives*, déclenchées uniquement lorsqu'elles sont invoquées explicitement à l'aide de l'instruction **notify**. L'extrait de code suivant montre comment le serveur Apache est redémarré uniquement par le gestionnaire **restart apache** lorsqu'un fichier de configuration est mis à jour et le notifie :

```
tasks:
  - name: copy demo.example.conf configuration template❶
    template:
      src: /var/lib/templates/demo.example.conf.template
      dest: /etc/httpd/conf.d/demo.example.conf
    notify: ❷
      - restart apache❸

handlers:
  - name: restart apache❹
    service: ❺
      name: httpd
```

```
state: restarted
```

- ➊ Tâche de notification du gestionnaire.
- ➋ L'argument **notify** indique que la tâche a besoin de déclencher un gestionnaire.
- ➌ Nom du gestionnaire à exécuter.
- ➍ Le mot-clé **handlers** indique le début de la liste des tâches du gestionnaire.
- ➎ Nom du gestionnaire invoqué par les tâches.
- ➏ Module à utiliser pour le gestionnaire.

Dans l'exemple précédent, le gestionnaire **restart apache** se déclenche une fois notifié par la tâche **template** qu'une modification a été apportée. Une tâche peut invoquer plusieurs gestionnaires dans la section **notify**. Ansible traite l'instruction **notify** comme une matrice et répète les noms des gestionnaires :

```
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
```

## UTILISATION DES GESTIONNAIRES

Comme indiqué dans la documentation Ansible, il convient de garder certaines choses importantes à l'esprit lors de l'utilisation de gestionnaires :

- Les gestionnaires fonctionnent toujours dans l'ordre spécifié par la section **handlers** du play. Ils ne s'exécutent pas dans l'ordre dans lequel ils sont listés par les instructions **notify** dans une tâche, ou dans l'ordre dans lequel les tâches les notifient.
- Les gestionnaires sont exécutés normalement lorsque toutes les autres tâches sont terminées. Un gestionnaire invoqué par une tâche dans la partie **tasks** d'un playbook n'est exécuté que lorsque *toutes* les tâches de la section **tasks** sont traitées. (Il y a quelques exceptions mineures à cela.)
- Les noms des gestionnaires existent dans un espace logique global. Si, par erreur, un même nom est attribué à deux gestionnaires différents, un seul gestionnaire sera exécuté.
- Même si plusieurs tâches notifient un gestionnaire, ce dernier ne sera exécuté qu'une seule fois. Si aucune tâche ne le notifie, le gestionnaire ne sera pas exécuté.

- Si une tâche qui inclut une instruction **notify** ne signale pas un résultat **changed** (par exemple, un paquetage est déjà installé et la tâche indique **ok**), le gestionnaire n'est pas notifié. Le gestionnaire est ignoré à moins qu'il soit notifié par une autre tâche. Ansible notifie les gestionnaires uniquement si la tâche signale l'état **changed**.



### IMPORTANT

Les gestionnaires sont censés effectuer une action supplémentaire lorsqu'une tâche modifie un hôte géré. Ils ne doivent pas être utilisés en remplacement des tâches normales.



### RÉFÉRENCES

#### Présentation des playbooks – Documentation Ansible

[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks\\_intro.html](https://docs.ansible.com/ansible/2.7/user_guide/playbooks_intro.html)

## ► EXERCICE GUIDÉ

# MISE EN ŒUVRE DES GESTIONNAIRES

Dans cet exercice, vous allez mettre en œuvre des gestionnaires dans des playbooks.

## RÉSULTATS

Vous devez pouvoir :

- Définir les gestionnaires dans des playbooks et les notifier des modifications apportées à la configuration.

Sur `workstation`, exécutez `lab control-handlers setup` pour configurer l'environnement pour l'exercice. Ce script crée le répertoire de projet `control-handlers`, et télécharge le fichier de configuration Ansible ainsi que le fichier d'inventaire des hôtes nécessaire pour cet exercice. Le répertoire du projet contient également un playbook partiellement complet, `configure_db.yml`.

```
[student@workstation ~]$ lab control-handlers setup
```

- 1. Sur `workstation.lab.example.com`, ouvrez un nouveau terminal et basculez vers le répertoire de projet `~/control-handlers`.

```
[student@workstation ~]$ cd ~/control-handlers
[student@workstation control-handlers]$
```

- 2. Dans ce répertoire, utilisez un éditeur de texte pour modifier le fichier de playbook `configure_db.yml`. Ce playbook va installer et configurer un serveur de base de données. Lorsque la configuration du serveur de base de données change, le playbook déclenche un redémarrage du service de base de données et configure le mot de passe administratif de la base de données.

- 2.1. À l'aide d'un éditeur de texte, passez en revue le playbook `configure_db.yml`. Cela commence par l'initialisation de certaines variables :

```
---
- name: MariaDB server is installed
hosts: databases
vars:
  db_packages:
    - mariadb-server
    - MySQL-python
  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:
```

- **db\_packages** : définit le nom des paquetages à installer pour le service de base de données.
  - **db\_service** : qui définit le nom du service de base de données.
  - **resources\_url** : URL du répertoire de ressources où se trouvent les fichiers de configuration distants.
  - **config\_file\_url** : URL du fichier de configuration de la base de données à installer.
  - **config\_file\_dst** : emplacement du fichier de configuration installé sur les hôtes gérés.
- 2.2. Dans le fichier **configure\_db.yml**, définissez une tâche qui utilise le module yum pour installer les paquetages de base de données requis, comme défini par la variable **db\_packages**. Si la tâche modifie le système, la base de données n'a pas été installée et vous devez en informer le gestionnaire **set db password** pour définir votre utilisateur de base de données et mot de passe initiaux. N'oubliez pas que la tâche du gestionnaire, si elle est notifiée, ne s'exécutera pas avant que chaque tâche de la section **tasks** se soit exécutée.

La tâche doit se présenter comme suit :

```
tasks:  
  - name: "{{ db_packages }} packages are installed"  
    yum:  
      name: "{{ db_packages }}"  
      state: present  
    notify:  
      - set db password
```

- 2.3. Ajoutez une tâche pour démarrer et activer le service de base de données. La tâche doit se présenter comme suit :

```
- name: Make sure the database service is running  
  service:  
    name: "{{ db_service }}"  
    state: started  
    enabled: true
```

- 2.4. Ajoutez une tâche pour télécharger **my.cnf.standard** vers **/etc/my.cnf** sur l'hôte géré, à l'aide du module `get_url`. Ajoutez une condition qui informe le gestionnaire **restart db service** de redémarrer le service de base de données après un changement de fichier de configuration. La tâche doit se présenter comme suit :

```
- 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. Ajoutez le mot-clé **handlers** pour définir le début des tâches du gestionnaire. Définissez le premier gestionnaire, **restart db service**, qui redémarre le service **mariadb**. Le contenu du fichier doit se présenter comme suit :

```
handlers:
  - name: restart db service
    service:
      name: "{{ db_service }}"
      state: restarted
```

- 2.6. Définissez le deuxième gestionnaire, **set db password**, qui définit le mot de passe administratif du service de base de données. Le gestionnaire utilise le module **mysql\_user** pour exécuter la commande. Le gestionnaire doit se présenter comme suit :

```
- name: set db password
  mysql_user:
    name: root
    password: redhat
```

Quand vous avez terminé, le playbook doit se présenter comme suit :

```
---
- name: MariaDB server is installed
  hosts: databases
  vars:
    db_packages:
      - mariadb-server
      - MySQL-python
    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. Avant d'exécuter le playbook, vérifiez que sa syntaxe est correcte en exécutant **ansible-playbook** avec l'option **--syntax-check**. En cas d'erreurs, corrigez-les avant de passer à l'étape suivante. Le résultat doit ressembler à ce qui suit :

```
[student@workstation control-handlers]$ ansible-playbook configure_db.yml \
> --syntax-check

playbook: configure_db.yml
```

- 4. Exécutez le playbook **configure\_db.yml**. Le résultat indique que les gestionnaires sont en cours d'exécution.

```
[student@workstation control-handlers]$ ansible-playbook configure_db.yml

PLAY [Installing MariaDB server] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [Install [u'mariadb-server', u'MySQL-python'] package] ****
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. Exécutez de nouveau le playbook.

```
[student@workstation control-handlers]$ ansible-playbook configure_db.yml

PLAY [Installing MariaDB server] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]

TASK [[u'mariadb-server', u'MySQL-python'] 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
```

Cette fois-ci, les gestionnaires sont ignorés. Si le fichier de configuration distant est modifié ultérieurement, l'exécution du playbook déclenchera le gestionnaire **restart db service** mais pas le gestionnaire **set db password**.

## Nettoyage

Sur workstation, exéutez le script **lab control-handlers cleanup** pour effacer les ressources créées au cours de cet exercice.

```
[student@workstation ~]$ lab control-handlers cleanup
```

L'exercice guidé est maintenant terminé.

# GESTION DES ÉCHECS DE TÂCHE

---

## OBJECTIFS

Après avoir terminé cette section, les stagiaires doivent pouvoir contrôler ce qui se passe lorsqu'une tâche échoue et quelles conditions provoquent son échec.

## ERREURS CONTENUES DANS LES PLAYS

Ansible évalue les codes de retour de chaque tâche pour déterminer si une tâche a réussi ou échoué. En règle générale, lorsqu'une tâche échoue, Ansible met immédiatement fin au reste du play sur cet hôte, en ignorant toutes les tâches suivantes.

Toutefois, dans certains cas, vous souhaiterez poursuivre l'exécution d'un play même si une tâche échoue. Par exemple, il se peut qu'une tâche particulière échoue, et vous pouvez souhaiter effectuer une récupération en exécutant une autre tâche de manière conditionnelle. Plusieurs fonctions Ansible peuvent être utilisées pour mieux gérer les erreurs liées aux tâches.

### Ignorance de l'échec d'une tâche

Par défaut, si une tâche échoue, le play est interrompu. Toutefois, vous pouvez modifier ce comportement en ignorant les tâches qui ont échoué. Vous pouvez utiliser le mot-clé **`ignore_errors`** dans une tâche pour accomplir cela.

L'extrait de code suivant montre comment utiliser **`ignore_errors`** dans une tâche pour poursuivre l'exécution d'un playbook sur l'hôte même en cas d'échec d'une tâche. Par exemple, si le paquetage *notapkg* n'existe pas, le module yum échoue, mais en définissant **`ignore_errors`** sur **`yes`**, l'exécution peut se poursuivre.

```
- name: Latest version of notapkg is installed
  yum:
    name: notapkg
    state: latest
    ignore_errors: yes
```

### Exécution forcée des gestionnaires après l'échec d'une tâche

Généralement, lorsqu'une tâche échoue et que le play est interrompu sur cet hôte, tout gestionnaire qui a été notifié par des tâches précédentes du play n'est pas exécuté. Si vous définissez le mot-clé **`force_handlers: yes`** sur le play, les gestionnaires notifiés sont appelés même si le play a été interrompu à cause de l'échec d'une tâche plus récente.

L'extrait de code suivant montre comment utiliser le mot-clé **`force_handlers`** dans un play pour forcer l'exécution du gestionnaire même si une tâche échoue :

```
---
- 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

```

**NOTE**

Gardez à l'esprit que les gestionnaires sont notifiés lorsqu'une tâche signale un résultat **changed**, mais ne le sont pas lorsque le résultat est **ok** ou **failed**.

## Spécification des conditions d'échec d'une tâche

Vous pouvez utiliser le mot-clé **failed\_when** sur une tâche pour spécifier les conditions indiquant que la tâche a échoué. Ceci est souvent utilisé avec des modules de commande pouvant exécuter une commande avec succès, mais le résultat de la commande indique un échec.

Vous pouvez par exemple exécuter un script générant un message d'erreur et utiliser ce message pour définir l'état d'échec de la tâche. L'extrait de code suivant montre comment utiliser le mot-clé **failed\_when** dans une tâche :

```

tasks:
- name: Run user creation script
  shell: /usr/local/bin/create_users.sh
  register: command_result
  failed_when: "'Password missing' in command_result.stdout"

```

Le module **fail** peut également être utilisé pour forcer un échec de tâche. Le scénario ci-dessus peut également être écrit sous la forme de deux tâches :

```

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"

```

Vous pouvez utiliser le module **fail** pour fournir un message d'échec clair pour la tâche. Cette approche permet également les échecs différés, vous permettant d'exécuter des tâches intermédiaires pour effectuer ou annuler d'autres modifications.

## Spécification d'une tâche signalant un résultat « changed »

Lorsqu'une tâche apporte une modification à un hôte géré, elle signale l'état **changed** et notifie les gestionnaires. Lorsqu'une tâche n'a besoin d'apporter aucune modification, elle signale l'état **ok** et ne notifie pas les gestionnaires.

Vous pouvez utiliser le mot-clé **changed\_when** pour contrôler le cas où une tâche signale une modification. Par exemple, le module **shell** figurant dans l'exemple suivant est chargé d'obtenir les informations d'identification Kerberos qui vont servir à des tâches ultérieures. Généralement, il signale toujours un état **changed** lorsqu'il est exécuté. Pour supprimer cette modification, **changed\_when: false** est défini pour signaler un état **ok** ou **failed**.

```
- name: get Kerberos credentials as "admin"
  shell: echo "{{ krb_admin_pass }}" | kinit -f admin
  changed_when: false
```

L'exemple suivant utilise le module **shell** pour signaler **changed** en fonction de la sortie du module collectée par une variable enregistrée :

```
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
```

## Blocs Ansible et gestion des erreurs

Dans les playbooks, les *blocs* sont des clauses qui regroupent des tâches de manière logique et qui peuvent être utilisées pour contrôler la façon dont les tâches sont exécutées. Par exemple, un bloc de tâche peut avoir un mot-clé **when** pour appliquer plusieurs tâches conditionnelles :

```
- name: block example
hosts: all
tasks:
- 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"
```

Les blocs permettent également la gestion des erreurs en association avec les instructions **rescue** et **always**. Si une tâche appartenant à un block échoue, les tâches figurant dans son bloc **rescue** sont exécutées dans l'ordre de récupération. Après l'exécution des tâches de la clause de bloc, ainsi que les tâches de la clause **rescue** en cas d'échec, les tâches de la clause **always** s'exécutent. En résumé :

- **block** : définit les tâches principales à exécuter.
- **rescue** : définit les tâches devant s'exécuter si les tâches définies dans la clause **block** échouent.
- **always** : définit les tâches qui seront systématiquement exécutées, indépendamment de la réussite ou de l'échec des tâches définies dans les clauses **block** et **rescue**.

L'exemple suivant indique comment mettre en œuvre un bloc dans un playbook. Même si les tâches définies dans la clause **block** échouent, celles définies dans les clauses **rescue** et **always** sont exécutées.

```
tasks:  
  - 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
```

La condition **when** sur une clause **block** s'applique également à ses clauses **rescue** et **always** le cas échéant.

Figure 5.0: Gestion des échecs de tâche



## RÉFÉRENCES

### Gestion des erreurs dans les playbooks – Documentation Ansible

[http://docs.ansible.com/ansible/playbooks\\_error\\_handling.html](http://docs.ansible.com/ansible/playbooks_error_handling.html)

### Gestion des erreurs – Blocs – Documentation Ansible

[http://docs.ansible.com/ansible/playbooks\\_blocks.html#error-handling](http://docs.ansible.com/ansible/playbooks_blocks.html#error-handling)

## ► EXERCICE GUIDÉ

# GESTION DES ÉCHECS DE TÂCHE

Dans cet exercice, vous allez explorer différentes manières de gérer l'échec d'une tâche dans un playbook Ansible.

## RÉSULTATS

Vous devez pouvoir :

- Ignorer les commandes ayant échoué lors de l'exécution des playbooks.
- Forcer l'exécution des gestionnaires.
- Ignorer ce qui constitue un échec dans les tâches.
- Ignorer l'état **changed** pour les tâches.
- Mettre en œuvre les clauses block/rescue/always dans des playbooks.

Sur `workstation`, exécutez le script de configuration de l'atelier pour vérifier que l'environnement est prêt pour commencer l'atelier. Ce script crée le répertoire de travail `~/control-errors`.

```
[student@workstation ~]$ lab control-errors setup
```

- 1. Sur `workstation.lab.example.com`, accédez au répertoire de projet `~/control-errors`.

```
[student@workstation ~]$ cd ~/control-errors
[student@workstation control-errors]$
```

- 2. Le script d'atelier créé dans un fichier de configuration Ansible, ainsi que dans un fichier d'inventaire contient le serveur `servera.lab.example.com` dans le groupe **databases**. Vérifiez le fichier avant de continuer.
- 3. Créez le playbook `playbook.yml` contenant un play avec deux tâches. Écrivez la première tâche contenant une erreur délibérée qui entraîne son échec.

- 3.1. Ouvrez le playbook dans un éditeur de texte. Définissez trois variables : `web_package` avec la valeur `http`, `db_package` avec la valeur `mariadb-server` et `db_service` avec la valeur `mariadb`. Les variables seront utilisées pour installer les paquetages requis et démarrer le serveur.

La valeur `http` est une erreur intentionnelle dans le nom du paquetage. Le fichier doit se présenter comme suit :

```
---
- name: Task Failure Exercise
```

```
hosts: databases
vars:
  web_package: http
  db_package: mariadb-server
  db_service: mariadb
```

- 3.2. Définissez deux tâches utilisant le module **yum** et les deux variables, **web\_package** et **db\_package**. La tâche va installer les paquetages requis. Les tâches doivent se présenter comme suit :

```
tasks:
  - name: Install {{ web_package }} package
    yum:
      name: "{{ web_package }}"
      state: present

  - name: Install {{ db_package }} package
    yum:
      name: "{{ db_package }}"
      state: present
```

- 4. Exéutez le playbook et observez la sortie du play.

```
[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, "msg": "No
package matching 'http' found available, installed or updated", "rc": 126,
"results": ...output omitted...}
to retry, use: --limit @/home/student/control-errors/playbook.retry

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

La tâche a échoué, car il n'existe aucun paquetage nommé **http**. La première tâche ayant échoué, la seconde n'a pas été exécutée.

- 5. Mettez la première tâche à jour afin d'ignorer les erreurs en ajoutant le mot-clé **ignore\_errors**. Les tâches doivent se présenter comme suit :

```
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. Exécutez le playbook à nouveau et observez la sortie du play.

```
[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, "msg": "No
 package matching 'http' found available, installed or updated", "rc": 126,
 "results": ...output omitted...}
...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
```

Bien que la première tâche ait échoué, Ansible a exécuté la seconde.

- 7. À cette étape, vous allez définir un mot-clé **block** pour pouvoir expérimenter son fonctionnement.

- 7.1. Mettez le playbook à jour en imbriquant la première tâche dans une clause **block**. Supprimez la ligne qui définit **ignore\_errors: yes**. Le bloc doit se présenter comme suit :

```
- block:
  - name: Install {{ web_package }} package
    yum:
      name: "{{ web_package }}"
      state: present
```

- 7.2. Imbriquez la tâche qui installe le paquetage *mariadb-server* dans une clause **rescue**. La tâche s'exécute si celle qui est listée dans la clause **block** échoue. La clause block doit se présenter comme suit :

```
rescue:
  - name: Install {{ db_package }} package
    yum:
      name: "{{ db_package }}"
```

```
state: present
```

- 7.3. Pour finir, ajoutez une clause **always** qui démarre le serveur de base de données une fois l'installation terminée à l'aide du module **service**. La clause doit se présenter comme suit :

```
always:
  - name: Start {{ db_service }} service
    service:
      name: "{{ db_service }}"
      state: started
```

- 7.4. La section de la tâche terminée doit se présenter comme suit :

```
tasks:
  - 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. Lancez à nouveau le playbook et observez le résultat.

- 8.1. Exécutez le playbook. La tâche du bloc qui vérifie l'installation de **web\_package** échoue, ce qui entraîne l'exécution de la tâche du bloc **rescue**. La tâche du bloc **always** est ensuite exécutée.

```
[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, "msg": "No package matching 'http' found available, installed or updated", "rc": 126, "results": [...output omitted...]}

TASK [Install mariadb-server package] ****
ok: [servera.lab.example.com]

TASK [Start mariadb service] ****
```

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

```
PLAY RECAP ****
servera.lab.example.com : ok=3    changed=1    unreachable=0    failed=1
```

- 8.2. Modifiez le playbook en remplaçant la valeur de la variable `web_package` par **httpd**. La tâche du bloc peut alors aboutir lors de la prochaine exécution du playbook.

```
vars:
  web_package: httpd
  db_package: mariadb-server
  db_service: mariadb
```

- 8.3. Exécutez de nouveau le playbook. Cette fois, la tâche du bloc n'échoue pas. Par conséquent, la tâche de la section **rescue** est ignorée. La tâche du bloc **always** continue d'être exécutée.

```
[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. Cette étape explore le moyen de contrôler la condition qui conduit une tâche être signalée comme « `changed` » à l'hôte géré.

- 9.1. Modifiez le playbook en ajoutant deux tâches au début du play, avant le **block**. La première tâche utilise le module **command** pour exécuter la commande **date** et enregistrer le résultat dans la variable **command\_result**. La seconde tâche utilise le module **debug** pour imprimer la sortie standard de la commande de la première tâche.

```
tasks:
  - name: Check local time
    command: date
    register: command_result

  - name: Print local time
    debug:
```

```
var: command_result.stdout
```

- 9.2. Exécutez le playbook. Vous devez constater que la première tâche exécutant le module **command** signale l'état **changed**, même si aucune modification n'a été apportée au système distant. Elle s'est contentée de collecter des informations sur l'heure. Cela est dû au fait que le module **command** ne peut pas faire la différence entre une commande qui collecte les données et une commande qui modifie l'état.

```
[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": "Fri Nov  9 15:30:39 EST 2018"
}

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
```

Si vous exécutez à nouveau le playbook, la tâche **Check local time** renvoie à nouveau **changed**.

- 9.3. Cette tâche **command** ne doit pas signaler l'état **changed** à chaque exécution, car elle n'apporte aucune modification à l'hôte géré. Sachant que la tâche ne modifie jamais un hôte géré, ajoutez la ligne **changed\_when: false** à la tâche pour supprimer la modification.

```
tasks:
  - name: Check local time
    command: date
    register: command_result
    changed_when: false

  - name: Print local time
    debug:
```

```
var: command_result.stdout
```

- 9.4. Exécutez le playbook une nouvelle fois pour constater que la tâche signale à présent l'état **ok**, mais la tâche est encore en cours d'exécution et continue d'enregistrer l'heure dans la variable.

```
[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": "Fri Nov  9 15:35:39 EST 2018"
}

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. Comme dernier exercice, modifiez le playbook pour explorer la façon dont le mot-clé **failed\_when** interagit avec les tâches.

- 10.1. Modifiez la tâche **Install {{ web\_package }} package** de sorte qu'elle signale un échec lorsque `web_package` a la valeur **httpd**. Comme c'est le cas, la tâche signale un échec lors de l'exécution du play.  
Effectuez soigneusement la mise en retrait de sorte que le mot-clé soit correctement défini sur la tâche.

```
- block:
  - name: Install {{ web_package }} package
    yum:
      name: "{{ web_package }}"
      state: present
    failed_when: web_package == "httpd"
```

- 10.2. Exécutez le playbook.

```
[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": "Fri Nov  9 15:40:37 EST 2018"
}

TASK [Install httpd package] ****
fatal: [servera.lab.example.com]: FAILED! => {"changed": false,
"failed_when_result": true, "msg": "", "rc": 0, "results":
["httpd-2.4.6-80.el7.x86_64 providing httpd is already installed"]}

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

```

Examinez attentivement la sortie. La tâche **Install httpd package** signale un échec, mais elle a été réellement exécutée et s'est assurée que le paquetage est installé en premier. Le mot-clé **failed\_when** modifie l'état signalé par la tâche après l'exécution de celle-ci, elle ne modifie pas le comportement de la tâche elle-même.

Toutefois, l'échec signalé peut modifier le comportement du reste du play. Dans la mesure où la tâche était dans un bloc et a signalé un échec, la tâche **Install mariadb-server package** dans la section **rescue** du bloc a été exécutée.

## Nettoyage

Sur **workstation**, exéutez le script **lab control-errors cleanup** pour effacer les ressources créées au cours de cet exercice.

```
[student@workstation ~]$ lab control-errors cleanup
```

L'exercice guidé est maintenant terminé.

## ► OPEN LAB

# MISE EN ŒUVRE D'UN CONTRÔLE DE TÂCHE

## LISTE DE CONTRÔLE DES PERFORMANCES

Dans cet atelier, vous allez installer le service Web Apache et le protéger à l'aide de mod\_ssl. Vous utiliserez des conditions, des gestionnaires et une gestion des échecs de tâches dans votre playbook pour déployer l'environnement.

## RÉSULTATS

Vous devez pouvoir définir des conditions dans des playbooks Ansible, configurer des boucles qui itèrent des éléments, définir des gestionnaires dans des playbooks et gérer les erreurs de tâches.

Connectez-vous en tant qu'utilisateur student sur `workstation` et exécutez `lab control-review setup`. Ce script de configuration garantit que l'hôte géré, `serverb`, est accessible sur le réseau. Il garantit également que le fichier de configuration Ansible et l'inventaire appropriés sont installés sur le nœud de contrôle.

```
[student@workstation ~]$ lab control-review setup
```

1. Sur `workstation.lab.example.com`, accédez au répertoire de projet `~/control-review`.
2. Le répertoire du projet contient également un playbook partiellement terminé, `playbook.yml`. A l'aide d'un éditeur de texte, ajoutez une tâche utilisant le module `fail` sous le commentaire `#Fail Fast Message`. Assurez-vous de fournir un nom approprié pour la tâche. Cette tâche ne doit être exécutée que lorsque le système distant ne répond pas à la configuration minimale requise.

La configuration minimale requise pour l'hôte distant est indiquée ci-dessous :

- Possède au moins la quantité de RAM spécifiée par la variable `min_ram_megabytes`. La variable `min_ram_megabytes` est définie dans le fichier `vars.yml` et a la valeur **256**.
- Exécute Red Hat Enterprise Linux.

3. Ajoutez une seule tâche au playbook sous le commentaire `#Install all Packages` pour installer les paquetages manquants. Les paquetages requis sont spécifiés par la variable `packages`, qui est définie dans le fichier `vars.yml`.

Le nom de la tâche devrait être `Ensure required packages are present`. Utilisez le module `package` pour installer des paquetages, pas le module `yum`.

4. Ajoutez une seule tâche au playbook sous le commentaire `#Enable and start services` pour démarrer tous les services. Tous les services spécifiés par la variable `services`, qui est définie dans le fichier `vars.yml`, doivent être démarrés et activés. Assurez-vous de fournir un nom approprié pour la tâche.

5. Ajoutez un bloc de tâches au playbook sous le commentaire `#Block of config tasks`. Ce bloc contient deux tâches :

- Une tâche pour s'assurer le répertoire spécifié par la variable `ssl_cert_dir` existe sur l'hôte distant. Ce répertoire stocke les certificats du serveur Web.
- Une tâche pour copier tous les fichiers spécifiés par la variable `web_config_files` sur l'hôte distant. Examinez la structure de la variable `web_config_files` dans le fichier `vars.yml`. Configurez la tâche pour copier chaque fichier vers la destination correcte sur l'hôte distant.

Cette tâche doit déclencher le gestionnaire `restart web service` si l'un de ces fichiers est modifié sur le serveur distant.

En outre, une tâche de débogage est exécutée si l'une des deux tâches ci-dessus échoue. Dans ce cas, la tâche imprime le message : **One or more of the configuration changes failed, but the web service is still active.** (Une ou plusieurs des modifications de configuration ont échoué, mais le service Web est toujours actif).

Assurez-vous de fournir un nom approprié pour toutes les tâches.

6. Le playbook configure l'hôte distant pour qu'il écoute les demandes HTTPS standard. Ajoutez une seule tâche au playbook sous le commentaire `#Configure the firewall` pour configurer `firewalld`.

Cette tâche doit garantir que l'hôte distant autorise les connexions HTTP et HTTPS standard. Ces modifications de configuration doivent être effectives immédiatement et persister après un redémarrage du système. Assurez-vous de fournir un nom approprié pour la tâche.

7. Définissez le gestionnaire `restart web service`.

Une fois déclenchée, cette tâche doit redémarrer le service Web défini par la variable `web_service`, définie dans le fichier `vars.yml`.

8. À partir du répertoire, `~/control-review`, exécutez le playbook `playbook.yml`. Le playbook doit s'exécuter sans erreur et déclencher l'exécution de la tâche du gestionnaire.

9. Vérifiez que le serveur Web répond maintenant aux demandes HTTPS en utilisant le certificat personnalisé auto-signé pour chiffrer la connexion. Le certificat personnalisé expire le 9 août 2021 et la réponse du serveur Web doit correspondre à la chaîne **Configured for both HTTP and HTTPS**.

## Évaluation

Sur `workstation`, exécutez la commande `lab control-review grade` pour confirmer que l'exercice est réussi. Corrigez toute erreur signalée et répétez le script tant que des erreurs persistent.

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

## Nettoyage

Exécutez la commande `lab control-review cleanup` pour procéder au nettoyage, une fois l'atelier terminé.

```
[student@workstation ~]$ lab control-review cleanup
```

## ► SOLUTION

# MISE EN ŒUVRE D'UN CONTRÔLE DE TÂCHE

### LISTE DE CONTRÔLE DES PERFORMANCES

Dans cet atelier, vous allez installer le service Web Apache et le protéger à l'aide de mod\_ssl. Vous utiliserez des conditions, des gestionnaires et une gestion des échecs de tâches dans votre playbook pour déployer l'environnement.

### RÉSULTATS

Vous devez pouvoir définir des conditions dans des playbooks Ansible, configurer des boucles qui itèrent des éléments, définir des gestionnaires dans des playbooks et gérer les erreurs de tâches.

Connectez-vous en tant qu'utilisateur student sur `workstation` et exécutez `lab control-review setup`. Ce script de configuration garantit que l'hôte géré, `serverb`, est accessible sur le réseau. Il garantit également que le fichier de configuration Ansible et l'inventaire appropriés sont installés sur le nœud de contrôle.

```
[student@workstation ~]$ lab control-review setup
```

1. Sur `workstation.lab.example.com`, accédez au répertoire de projet `~/control-review`.

```
[student@workstation ~]$ cd ~/control-review
[student@workstation control-review]$
```

2. Le répertoire du projet contient également un playbook partiellement terminé, `playbook.yml`. A l'aide d'un éditeur de texte, ajoutez une tâche utilisant le module `fail` sous le commentaire `#Fail Fast Message`. Assurez-vous de fournir un nom approprié pour la tâche. Cette tâche ne doit être exécutée que lorsque le système distant ne répond pas à la configuration minimale requise.

La configuration minimale requise pour l'hôte distant est indiquée ci-dessous :

- Possède au moins la quantité de RAM spécifiée par la variable `min_ram_megabytes`. La variable `min_ram_megabytes` est définie dans le fichier `vars.yml` et a la valeur **256**.
- Exécute Red Hat Enterprise Linux.

La tâche complétée correspond à :

```
tasks:
  #Fail Fast Message
  - name: Show Failed System Requirements Message
    fail:
      msg: "The {{ inventory_hostname }} did not meet minimum reqs."
```

```
when: >
  ansible_memtotal_mb*1024*1024 < min_ram_megabytes*1000000 or
  ansible_distribution != "RedHat"
```

3. Ajoutez une seule tâche au playbook sous le commentaire **#Install all Packages** pour installer les paquetages manquants. Les paquetages requis sont spécifiés par la variable **packages**, qui est définie dans le fichier **vars.yml**.

Le nom de la tâche devrait être **Ensure required packages are present**. Utilisez le module package pour installer des paquetages, pas le module yum.

La tâche complétée correspond à :

```
#Install all Packages
- name: Ensure required packages are present
  package:
    name: "{{ item }}"
    state: present
  loop: "{{ packages }}"
```

4. Ajoutez une seule tâche au playbook sous le commentaire **#Enable and start services** pour démarrer tous les services. Tous les services spécifiés par la variable **services**, qui est définie dans le fichier **vars.yml**, doivent être démarrés et activés. Assurez-vous de fournir un nom approprié pour la tâche.

La tâche complétée correspond à :

```
#Enable and start services
- name: Ensure services are started and enabled
  service:
    name: "{{ item }}"
    state: started
    enabled: yes
  loop: "{{ services }}"
```

5. Ajoutez un bloc de tâches au playbook sous le commentaire **#Block of config tasks**. Ce bloc contient deux tâches :

- Une tâche pour s'assurer le répertoire spécifié par la variable **ssl\_cert\_dir** existe sur l'hôte distant. Ce répertoire stocke les certificats du serveur Web.
- Une tâche pour copier tous les fichiers spécifiés par la variable **web\_config\_files** sur l'hôte distant. Examinez la structure de la variable **web\_config\_files** dans le fichier **vars.yml**. Configurez la tâche pour copier chaque fichier vers la destination correcte sur l'hôte distant.

Cette tâche doit déclencher le gestionnaire **restart web service** si l'un de ces fichiers est modifié sur le serveur distant.

En outre, une tâche de débogage est exécutée si l'une des deux tâches ci-dessus échoue. Dans ce cas, la tâche imprime le message : **One or more of the configuration**

**changes failed, but the web service is still active.** (Une ou plusieurs des modifications de configuration ont échoué, mais le service Web est toujours actif).

Assurez-vous de fournir un nom approprié pour toutes les tâches.

Le bloc de tâches complété correspond à :

```
#Block of config tasks
- 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. Le playbook configure l'hôte distant pour qu'il écoute les demandes HTTPS standard. Ajoutez une seule tâche au playbook sous le commentaire **#Configure the firewall** pour configurer firewalld.

Cette tâche doit garantir que l'hôte distant autorise les connexions HTTP et HTTPS standard. Ces modifications de configuration doivent être effectives immédiatement et persister après un redémarrage du système. Assurez-vous de fournir un nom approprié pour la tâche.

La tâche complétée correspond à :

```
#Configure the firewall
- name: ensure web server ports are open
  firewalld:
    service: "{{ item }}"
    immediate: true
    permanent: true
    state: enabled
  loop:
    - http
    - https
```

**7. Définissez le gestionnaire `restart web service`.**

Une fois déclenchée, cette tâche doit redémarrer le service Web défini par la variable `web_service`, définie dans le fichier `vars.yml`.

Une section `handlers` est ajoutée à la fin du playbook :

```
handlers:
  - name: restart web service
    service:
      name: "{{ web_service }}"
      state: restarted
```

Le playbook terminé contient les éléments suivants :

```
---
- 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*1024*1024 < min_ram_megabytes*1000000 or
      ansible_distribution != "RedHat"

  #Install all Packages
  - name: Ensure required packages are present
    package:
      name: "{{ item }}"
      state: present
    loop: "{{ packages }}"

  #Enable and start services
  - name: Ensure services are started and enabled
    service:
      name: "{{ item }}"
      state: started
      enabled: yes
    loop: "{{ services }}"

  #Block of config tasks
  - 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. À partir du répertoire, `~/control-review`, exécutez le playbook `playbook.yml`. Le playbook doit s'exécuter sans erreur et déclencher l'exécution de la tâche du gestionnaire.

```

[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] => (item=httpd)
changed: [serverb.lab.example.com] => (item=mod_ssl)
ok: [serverb.lab.example.com] => (item=firewalld)

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={'dest': '/etc/httpd/conf.d/ssl',
  'src': 'server.key'})
changed: [serverb.lab.example.com] => (item={'dest': '/etc/httpd/conf.d/ssl',
  'src': 'server.crt'})
changed: [serverb.lab.example.com] => (item={'dest': '/etc/httpd/conf.d',
  'src': 'ssl.conf'})
changed: [serverb.lab.example.com] => (item={'dest': '/var/www/html', 'src':
  'index.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

```

- Vérifiez que le serveur Web répond maintenant aux demandes HTTPS en utilisant le certificat personnalisé auto-signé pour chiffrer la connexion. Le certificat personnalisé expire le 9 août 2021 et la réponse du serveur Web doit correspondre à la chaîne **Configured for both HTTP and HTTPS**.

```

[student@workstation control-review]$ curl -k -vvv 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

```

## Évaluation

Sur `workstation`, exécutez la commande `lab control-review grade` pour confirmer que l'exercice est réussi. Corrigez toute erreur signalée et répétez le script tant que des erreurs persistent.

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

## Nettoyage

Exécutez la commande **lab control-review cleanup** pour procéder au nettoyage, une fois l'atelier terminé.

```
[student@workstation ~]$ lab control-review cleanup
```

# RÉSUMÉ

---

Dans ce chapitre, vous avez appris les principes suivants :

- Les boucles sont utilisées pour itérer un ensemble de valeurs, une liste simple de chaînes ou une liste de hachages/dictionnaires.
- Les conditions permettent d'exécuter des tâches ou des playbooks uniquement lorsque certaines conditions sont remplies.
- Les conditions sont testées avec plusieurs opérateurs, y compris des comparaisons de chaînes, des opérateurs mathématiques et des valeurs booléennes.
- Les gestionnaires correspondent à des tâches qui s'exécutent à la fin d'un playbook s'ils sont notifiés par d'autres tâches.
- Les gestionnaires ne sont notifiés que lorsqu'une tâche indique avoir modifié quelque chose sur un hôte géré.
- Les tâches sont configurées pour gérer des conditions d'erreur en ignorant les échecs, en forçant l'invocation des gestionnaires même si la tâche échoue, pour marquer une tâche comme ayant échoué lorsqu'elle réussit, ou remplacer le comportement qui fait qu'une tâche est marquée comme étant modifiée.
- Les blocs permettent de regrouper des tâches en une unité et d'exécuter d'autres tâches si toutes les tâches du bloc aboutissent.



## CHAPITRE 6

# DÉPLOIEMENT DE FICHIERS SUR DES HÔTES GÉRÉS

### PROJET

Déployer, gérer et ajuster les fichiers sur des hôtes gérés par Ansible.

### OBJECTIFS

- Créer, installer, modifier et supprimer des fichiers sur des hôtes gérés, et gérer les autorisations, la propriété, le contexte SELinux et d'autres caractéristiques de ces fichiers.
- Déployer des fichiers sur des hôtes gérés personnalisés à l'aide de modèles Jinja2.

### SECTIONS

- Modification et copie de fichiers sur des hôtes (et exercice guidé)
- Déploiement de fichiers personnalisés avec des modèles Jinja2 (et exercice guidé)

### ATELIER

- Déploiement de fichiers sur des hôtes gérés

# MODIFICATION ET COPIE DE FICHIERS SUR DES HÔTES

---

## OBJECTIFS

Au terme de cette section, les stagiaires doivent pouvoir créer, installer, modifier et supprimer des fichiers sur des hôtes gérés, ainsi que gérer les autorisations, la propriété, le contexte SELinux et d'autres caractéristiques de ces fichiers.

## DESCRIPTION DES MODULES DE FICHIERS

Red Hat Ansible Engine est livré avec une grande collection de modules (la « bibliothèque de modules ») qui sont développés dans le cadre du projet Ansible en amont. Pour faciliter leur organisation, leur documentation et leur gestion, ils sont organisés en groupes selon leur fonction de la documentation et lorsqu'ils sont installés sur un système.

La bibliothèque de modules `Files` inclut des modules qui vous permettent d'accomplir la plupart des tâches liées à la gestion de fichiers Linux, telles que la création, la copie, l'édition et la modification des autorisations et autres attributs des fichiers. Le tableau suivant fournit une liste des modules de gestion de fichiers fréquemment utilisés :

### Modules de fichiers couramment utilisés

| NOM DU MODULE            | DESCRIPTION DU MODULE                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>blockinfile</code> | Insérez, mettez à jour ou supprimez un bloc de texte multiligne entouré de lignes de marqueur personnalisables.                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <code>copy</code>        | Copiez un fichier depuis la machine locale ou distante vers un emplacement sur un hôte géré. Semblable au module <code>file</code> , le module <code>copy</code> peut également définir des attributs de fichier, y compris le contexte SELinux.                                                                                                                                                                                                                                                                                   |
| <code>fetch</code>       | Ce module fonctionne comme le module <code>copy</code> , mais en sens inverse. Ce module est utilisé pour récupérer des fichiers depuis des machines distantes vers le nœud de contrôle et les stocker dans une arborescence de fichiers, organisée par nom d'hôte.                                                                                                                                                                                                                                                                |
| <code>file</code>        | Définissez des attributs tels que les autorisations, la propriété, les contextes SELinux et les horodatages des fichiers normaux, des liens symboliques, des liens physiques et des répertoires. Ce module peut également créer ou supprimer des fichiers normaux, des liens symboliques, des liens physiques et des répertoires. Un certain nombre d'autres modules liés aux fichiers prennent en charge les mêmes options pour définir des attributs comme le module <code>file</code> , y compris le module <code>copy</code> . |
| <code>lineinfile</code>  | Assurez-vous qu'une ligne particulière se trouve dans un fichier ou remplacez une ligne existante à l'aide d'une expression régulière de repère arrière. Ce module est principalement utile lorsque vous souhaitez modifier une seule ligne dans un fichier.                                                                                                                                                                                                                                                                       |

| NOM DU MODULE | DESCRIPTION DU MODULE                                                                                                                                                                                                                                                                                                                                                                                                                            |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| stat          | Récupérez des informations sur l'état d'un fichier, similaire à la commande <b>stat</b> de Linux.                                                                                                                                                                                                                                                                                                                                                |
| synchronize   | Une enveloppe autour du <b>rsync</b> commande pour rendre les tâches courantes rapides et faciles. Le module <b>synchronize</b> n'est pas destiné à donner accès à toute la puissance de la commande <b>rsync</b> , mais il facilite la mise en œuvre des invocations les plus courantes. Vous devrez peut-être quand même appeler la commande <b>rsync</b> directement via un module <b>run</b> command en fonction de votre cas d'utilisation. |

## EXEMPLES D'AUTOMATISATION AVEC DES MODULES DE FICHIERS

La création, la copie, la modification et la suppression de fichiers sur des hôtes gérés sont des tâches courantes que vous pouvez implémenter à l'aide de modules provenant de bibliothèque de modules **Files**. Les exemples suivants montrent comment utiliser ces modules pour automatiser les tâches courantes de gestion de fichiers.

### Vérification de l'existence d'un fichier sur les hôtes gérés

Utilisez le module **file** pour toucher un fichier sur des hôtes gérés. Cela fonctionne comme la commande **touch**, en créant un fichier vide s'il n'existe pas et en mettant à jour son heure de modification s'il existe. Dans cet exemple, en plus de toucher le fichier, Ansible s'assure que l'utilisateur, le groupe et les autorisations du fichier sont définis sur des valeurs spécifiques.

```
- name: Touch a file and set permissions
  file:
    path: /path/to/file
    owner: user1
    group: group1
    mode: 0640
    state: touch
```

Exemple de résultat :

```
$ ls -l file
-rw-r-----  user1 group1 0 Nov 25 08:00 file
```

### Modification des attributs de fichiers

Vous pouvez utiliser le module **file** pour vous assurer qu'un fichier nouveau ou existant dispose des autorisations appropriées ou du type SELinux également.

Par exemple, le fichier suivant a conservé le contexte SELinux par défaut relatif au répertoire personnel d'un utilisateur, ce qui n'est pas le contexte souhaité.

```
$ ls -Z samba_file
-rw-r--r-- owner group unconfined_u:object_r:user_home_t:s0 samba_file
```

La tâche suivante permet de s'assurer que l'attribut de type de contexte SELinux du fichier **samba\_file** est du type **samba\_share\_t** souhaité. Ce comportement est similaire à la commande **chcon** Linux.

```
- name: SELinux type is set to samba_share_t
  file:
    path: /path/to/samba_file
    setype: samba_share_t
```

Exemple de résultat :

```
$ ls -Z samba_file
-rw-r--r-- owner group unconfined_u:object_r:samba_share_t:s0 samba_file
```

**NOTE**

Les paramètres d'attribut de fichier sont disponibles dans plusieurs modules de gestion de fichiers. Exécutez les commandes **ansible-doc file** et **ansible-doc copy** pour des informations supplémentaires.

## Rendre les modifications de contexte de fichier SELinux persistantes

Le module **file** agit comme **chcon** lors de la définition des contextes de fichier. Les modifications apportées avec ce module pourraient être annulées de manière inattendue en exécutant **restorecon**. Après avoir utilisé **file** pour définir le contexte, vous pouvez utiliser **sefcontext** de la collection de modules **System** pour mettre à jour la politique SELinux comme **semanage fcontext**.

```
- name: SELinux type is persistently set to samba_share_t
  sefcontext:
    target: /path/to/samba_file
    setype: samba_share_t
    state: present
```

Exemple de résultat :

```
$ ls -Z samba_file
-rw-r--r-- owner group unconfined_u:object_r:samba_share_t:s0 samba_file
```

**IMPORTANT**

Le module **sefcontext** met à jour le contexte par défaut de la cible dans la politique SELinux, mais ne modifie pas le contexte sur les fichiers existants.

## Copie et modification de fichiers sur des hôtes gérés

Dans cet exemple, le module **copy** est utilisé pour copier un fichier situé dans le répertoire de travail Ansible sur le nœud de contrôle vers les hôtes gérés sélectionnés.

Par défaut, ce module suppose que **force: yes** est défini. Cela oblige le module à écraser le fichier distant s'il existe mais qu'il contient un contenu différent du fichier en cours de copie. Si **force: no** est défini, il ne copie le fichier sur l'hôte géré que s'il n'existe pas déjà.

```
- name: Copy a file to managed hosts
  copy:
    src: file
    dest: /path/to/file
```

Pour vous assurer qu'une seule ligne de texte existe dans un fichier existant, utilisez le module **lineinfile**:

```
- name: Add a line of text to a file
  lineinfile:
    path: /path/to/file
    line: 'Add this line to the file'
    state: present
```

Pour ajouter un bloc de texte à un fichier existant, utilisez le module **blockinfile**:

```
- 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
```



#### NOTE

Lorsque vous utilisez le module **blockinfile**, les marqueurs de blocs commentés sont insérés au début et à la fin du bloc pour assurer l'idempotence.

```
# BEGIN ANSIBLE MANAGED BLOCK
First line in the additional block of text
Second line in the additional block of text
# END ANSIBLE MANAGED BLOCK
```

Vous pouvez utiliser le paramètre **marker** du module pour vous assurer que le caractère ou le texte de commentaire correct est utilisé pour le fichier en question.

## Suppression d'un fichier sur des hôtes gérés

Un exemple de base pour supprimer un fichier sur des hôtes gérés consiste à utiliser le module **file** avec le paramètre **state: absent**. Le paramètre **state** est facultatif pour de nombreux modules. Vous devez toujours préciser si vous voulez **state: present** ou **state: absent** pour plusieurs raisons. Certains modules prennent également en charge d'autres options. Il est possible que la valeur par défaut puisse changer à un moment donné, mais peut-être plus important encore, cela permet de comprendre plus facilement l'état dans lequel le système devrait être en fonction de votre tâche.

```
- name: Make sure a file does not exist on managed hosts
  file:
    dest: /path/to/file
    state: absent
```

## Récupération de l'état d'un fichier sur des hôtes gérés

Le module `stat` récupère des faits pour un fichier, similaire à la commande `stat` de Linux. Les paramètres fournissent la fonctionnalité permettant de récupérer les attributs de fichier, de déterminer la somme de contrôle d'un fichier, etc.

Le module `stat` retourne un hachage/dictionnaire de valeurs contenant les données d'état du fichier, ce qui vous permet de faire référence à des informations individuelles à l'aide de variables distinctes.

L'exemple suivant enregistre les résultats d'un module `stat` puis imprime la somme de contrôle MD5 du fichier qu'il a vérifié. (L'algorithme SHA256, plus moderne, est également disponible ; MD5 est utilisé ici pour rendre la sortie d'exemple plus lisible.)

```
- 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 }}"
```

Le résultat doit ressembler à ce qui suit :

```
TASK [Get md5 checksum of a file] ****
ok: [hostname]

TASK [debug] ****
ok: [hostname] => {
  "msg": "The checksum of the file is 5f76590425303022e933c43a7f2092a3"
}
```

Les informations sur les valeurs renvoyées par le module `stat` sont documentées par [ansible-doc](#), ou vous pouvez enregistrer une variable et afficher son contenu pour voir ce qui est disponible :

```
- 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
```

## Synchronisation de fichiers entre le nœud de contrôle et les hôtes gérés

Le module `synchronize` est un wrapper autour de l'outil `rsync`, qui simplifie les tâches courantes de gestion de fichiers dans vos playbooks. L'outil `rsync` doit être installé à la fois sur l'hôte local et sur l'hôte distant. Par défaut, lorsque vous utilisez le module `synchronize`, « l'hôte local » est l'hôte d'origine de la tâche de synchronisation (généralement, le nœud de contrôle), et l'« hôte de destination » est l'hôte auquel `synchronize` se connecte.

L'exemple suivant synchronise un fichier situé dans le répertoire de travail Ansible avec les hôtes gérés :

```
- name: synchronize local file to remote files
  synchronize:
    src: file
    dest: /path/to/file
```

Il y a plusieurs façons d'utiliser le module `synchronize` et ses nombreux paramètres, y compris la synchronisation des répertoires. Exécutez la commande `ansible-doc synchronize` pour accéder à des paramètres et des exemples de playbook supplémentaires.



### RÉFÉRENCES

Pages du manuel `ansible-doc(1)`, `chmod(1)`, `chown(1)`, `rsync(1)`, `stat(1)` et `touch(1)`

### Modules de fichiers

[https://docs.ansible.com/ansible/2.7/modules/list\\_of\\_files\\_modules.html](https://docs.ansible.com/ansible/2.7/modules/list_of_files_modules.html)

## ► EXERCICE GUIDÉ

# MODIFICATION ET COPIE DE FICHIERS SUR DES HÔTES

Dans cet exercice, vous allez utiliser des modules Ansible standard pour créer, installer, modifier et supprimer des fichiers sur des hôtes gérés, et gérer les autorisations, la propriété et les contextes SELinux de ces fichiers.

## RÉSULTATS

Vous devez pouvoir réaliser les tâches suivantes :

- Récupérez les fichiers à partir des hôtes gérés et stockez-les localement par nom d'hôte.
- Créez des playbooks utilisant des modules de gestion de fichiers courants tels que `copy`, `file`, `lineinfile` et `blockinfile`.

Sur `workstation`, exéutez le script `lab file-manage setup` afin de configurer l'environnement pour cet exercice. Ce script crée le répertoire de projet `file-manage`, et télécharge le fichier de configuration Ansible ainsi que le fichier d'inventaire des hôtes nécessaire pour cet exercice.

```
[student@workstation ~]$ lab file-manage setup
```

- 1. En tant qu'utilisateur `student` sur `workstation`, accédez au répertoire de travail `/home/student/file-manage`.

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

- 2. Créez un playbook appelé `secure_log_backups.yml` dans le répertoire de travail actuel. Configurez le playbook pour utiliser le module `fetch` afin de récupérer le fichier journal `/var/log/secure` de chacun des hôtes gérés et les stocker sur le nœud de contrôle. Le playbook doit créer le répertoire `secure_log_backups` avec des sous-répertoires nommés d'après le nom d'hôte de chaque hôte géré. Stockez les fichiers de sauvegarde dans leurs sous-répertoires respectifs.

- 2.1. Créez le playbook `secure_log_backups.yml` avec le contenu initial :

```
---
- name: Use the fetch module to retrieve secure log files
  hosts: all
  remote_user: root
```

- 2.2. Ajoutez une tâche au playbook `secure_log_backups.yml` qui récupère le fichier journal `/var/log/secure` à partir des hôtes gérés et le stocke dans le répertoire

**~/file-manage/secure-backups.** Si le répertoire **~/file-manage/secure-backups** n'existe pas, il est créé automatiquement par le module `fetch`. Utilisez le paramètre `flat: no` pour assurer le comportement par défaut de l'ajout du nom d'hôte, du chemin d'accès et du nom de fichier à la destination :

```
tasks:
  - name: Fetch the /var/log/secure log file from managed hosts
    fetch:
      src: /var/log/secure
      dest: secure-backups
      flat: no
```

- 2.3. Avant d'exécuter le playbook, exécutez la commande **ansible-playbook --syntax-check secure\_log\_backups.yml** pour vérifier sa syntaxe. En cas d'erreurs, corrigez-les avant de passer à l'étape suivante.

```
[student@workstation file-manage]$ ansible-playbook --syntax-check \
> secure_log_backups.yml

playbook: secure_log_backups.yml
[student@workstation file-manage]$
```

- 2.4. Exécutez **ansible-playbook secure\_log\_backups.yml** pour lancer le playbook.

```
[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
```

- 2.5. Vérifiez les résultats du playbook :

```
[student@workstation file-manage]$ tree -F -P secure
...output omitted...
└── secure-backups/
    ├── servera.lab.example.com/
    │   └── var/
    │       └── log/
    │           └── secure
    └── serverb.lab.example.com/
        └── var/
            └── log/
```

```

    └── secure
...output omitted...

```

- 3. Créez le playbook **copy\_file.yml** dans le répertoire de travail actuel. Configurez le playbook pour copier le fichier **/home/student/file-manage/files/users.txt** sur les hôtes gérés.

3.1. Ajoutez le contenu initial suivant au playbook **copy\_file.yml**:

#### Paramètres

| PARAMÈTRE   | VALEURS                               |
|-------------|---------------------------------------|
| name        | <b>Utilisation du module de copie</b> |
| hosts       | <b>all</b>                            |
| remote_user | <b>root</b>                           |

```

---
- name: Using the copy module
  hosts: all
  remote_user: root

```

- 3.2. Ajoutez une tâche pour utiliser le module copy afin de copier le fichier **/home/student/file-manage/files/users.txt** sur tous les hôtes gérés. Utilisez le module copy afin de définir les paramètres suivants pour le fichier **users.txt**:

#### Paramètres

| PARAMÈTRE | VALEURS                       |
|-----------|-------------------------------|
| src       | <b>files/users.txt</b>        |
| dest      | <b>/home/devops/users.txt</b> |
| owner     | <b>devops</b>                 |
| group     | <b>devops</b>                 |
| mode      | <b>u+rwx, g-wx, o-rwx</b>     |
| setype    | <b>samba_share_t</b>          |

```

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
```

- 3.3. Utilisez la commande **ansible-playbook --syntax-check copy\_file.yml** pour vérifier la syntaxe du playbook **copy\_file.yml**.

```
[student@workstation file-manage]$ ansible-playbook --syntax-check copy_file.yml

playbook: copy_file.yml
```

- 3.4. Exécutez le playbook :

```
[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
```

- 3.5. Utilisez une commande ad hoc pour exécuter la commande **ls -Z** en tant qu'utilisateur **devops** afin de vérifier les attributs du fichier **users.txt** sur les hôtes gérés.

```
[student@workstation file-manage]$ ansible all -m command -a 'ls -Z' -u devops
servera.lab.example.com | CHANGED | rc=0 >>
-rw-r-----. devops devops unconfined_u:object_r:samba_share_t:s0 users.txt

serverb.lab.example.com | CHANGED | rc=0 >>
-rw-r-----. devops devops unconfined_u:object_r:samba_share_t:s0 users.txt
```

- 4. Dans une étape précédente, le champ de type SELinux **samba\_share\_t** a été défini pour le fichier **users.txt**. Cependant, il est maintenant déterminé que les valeurs par défaut doivent être définies pour le contexte du fichier SELinux.

Créez un playbook appelé **selinux\_defaults.yml** dans le répertoire de travail actuel. Configurez le playbook de façon à utiliser le module **file** pour garantir le contexte SELinux par défaut pour les champs **user**, **role**, **type** et **level**.

- 4.1. Créez le 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
```

- 4.2. Utilisez la commande **ansible-playbook --syntax-check** **selinux\_defaults.yml** pour vérifier la syntaxe du playbook **selinux\_defaults.yml**.

```
[student@workstation file-manage]$ ansible-playbook --syntax-check \
> selinux_defaults.yml

playbook: selinux_defaults.yml
```

- 4.3. Exécutez le playbook :

```
[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
```

- 4.4. Utilisez une commande ad hoc pour exécuter la commande **ls -Z** en tant qu'utilisateur devops afin de vérifier les attributs du fichier par défaut de **unconfined\_u:object\_r:user\_home\_t:s0**.

```
[student@workstation file-manage]$ ansible all -m command -a 'ls -Z' -u devops
servera.lab.example.com | CHANGED | rc=0 >>
-rw-r-----. devops devops unconfined_u:object_r:user_home_t:s0 users.txt

serverb.lab.example.com | CHANGED | rc=0 >>
-rw-r-----. devops devops unconfined_u:object_r:user_home_t:s0 users.txt
```

- 5. Créez un playbook appelé **add\_line.yml** dans le répertoire de travail actuel. Configurez le playbook pour utiliser le module **lineinfile** afin d'ajouter la ligne **This line was**

**added by the lineinfile module** au fichier **/home/devops/users.txt** sur tous les hôtes gérés.

5.1. Créez le 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
```

5.2. Utilisez la commande **ansible-playbook --syntax-check add\_line.yml** pour vérifier la syntaxe du playbook **add\_line.yml**.

```
[student@workstation file-manage]$ ansible-playbook --syntax-check add_line.yml
playbook: add_line.yml
```

5.3. Exéutez le playbook:

```
[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
```

5.4. Utilisez le module **command** avec l'option **cat**, en tant qu'utilisateur **devops**, pour vérifier le contenu du fichier **users.txt** sur les hôtes gérés.

```
[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.
```

- 6. Créez un playbook appelé **add\_block.yml** dans le répertoire de travail actuel. Configurez le playbook pour utiliser le module `blockinfile` afin d'ajouter le bloc de texte suivant au fichier `/home/devops/users.txt` sur tous les hôtes gérés.

```
This block of text consists of two lines.  
They have been added by the blockinfile module.
```

- 6.1. Créez le 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
```

- 6.2. Utilisez la commande **ansible-playbook --syntax-check add\_block.yml** pour vérifier la syntaxe du playbook **add\_block.yml**.

```
[student@workstation file-manage]$ ansible-playbook --syntax-check add_block.yml  
playbook: add_block.yml
```

- 6.3. Exécutez le playbook:

```
[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
```

- 6.4. Utilisez le module `command` avec la commande `cat` pour vérifier le contenu correct du fichier `/home/devops/users.txt` sur l'hôte géré.

```
[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
```

- ▶ 7. Créez un playbook appelé **remove\_file.yml** dans le répertoire de travail actuel. Configurez le playbook pour utiliser le module `file` afin de supprimer le fichier `/home/devops/users.txt` de tous les hôtes gérés.

7.1. Créez le 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
```

7.2. Utilisez la commande **ansible-playbook --syntax-check remove\_file.yml** pour vérifier la syntaxe du playbook **remove\_file.yml**.

```
[student@workstation file-manage]$ ansible-playbook --syntax-check remove_file.yml
playbook: remove_file.yml
```

7.3. Exécutez le playbook:

```
[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
```

- 7.4. Utilisez une commande ad hoc pour exécuter la commande **ls -l** pour confirmer que le fichier **users.txt** n'existe plus sur les hôtes gérés.

```
[student@workstation file-manage]$ ansible all -m command -a 'ls -l'
serverb.lab.example.com | CHANGED | rc=0 >>
total 0

servera.lab.example.com | CHANGED | rc=0 >>
total 0
```

## Nettoyage

À partir de **workstation**, exécutez le script **lab file-manage cleanup** pour effacer cet exercice.

```
[student@workstation ~]$ lab file-manage cleanup
```

L'exercice guidé est maintenant terminé.

# DÉPLOIEMENT DE FICHIERS PERSONNALISÉS AVEC DES MODÈLES JINJA2

---

## OBJECTIFS

Au terme de cette, les stagiaires doivent pouvoir déployer des fichiers sur des hôtes gérés personnalisés à l'aide de modèles Jinja2.

## CRÉATION DE MODÈLES À PARTIR DE FICHIERS

Red Hat Ansible Engine dispose d'un certain nombre de modules qui peuvent être utilisés pour modifier des fichiers existants. Ceux-ci incluent `lineinfile` et `blockinfile`, entre autres. Cependant, ils ne sont pas toujours faciles à utiliser efficacement et correctement.

Un moyen beaucoup plus puissant de gérer les fichiers est de les *transformer en modèles*. Avec cette méthode, vous pouvez écrire un fichier de configuration de modèle personnalisé automatiquement pour l'hôte géré lors du déploiement du fichier, en utilisant des variables et des faits Ansible. Cela peut être plus facile à contrôler et générer moins d'erreurs.

## PRÉSENTATION DE JINJA2

Ansible utilise le système de création de modèles Jinja2 pour les fichiers modèles. Ansible utilise également la syntaxe Jinja2 pour référencer les variables dans les playbooks, vous savez donc déjà un peu comment l'utiliser.

### Utilisation de délimiteurs

Les variables et les expressions logiques sont placées entre balises, ou délimiteurs. Par exemple, les modèles Jinja2 utilisent `{% EXPR %}` pour les expressions ou la logique (des boucles, par exemple), tandis que la balise `{{ EXPR }}` est utilisée pour générer les résultats d'une expression ou d'une variable pour l'utilisateur final. Cette dernière, une fois le rendu effectué, est remplacée par une ou plusieurs valeurs et visualisée par l'utilisateur final. Utilisez la syntaxe `{# COMMENT #}` pour inclure des commentaires qui ne doivent pas apparaître dans le fichier final.

La première ligne de l'exemple suivant contient un commentaire qui ne sera pas inclus dans le fichier final. Les références de variable de la deuxième ligne sont remplacées par les valeurs des faits système auxquels il est fait référence.

```
{# /etc/hosts line #}
{{ ansible_facts.default_ipv4.address }}    {{ ansible_facts.hostname }}
```

## GÉNÉRATION D'UN MODÈLE JINJA2

Un modèle Jinja2 se compose de plusieurs éléments : des données, des variables et des expressions. Ces variables et expressions sont remplacées par leurs valeurs lors du rendu du modèle Jinja2. Les variables utilisées dans le modèle peuvent être spécifiées dans la section `vars` du playbook. Il est possible d'utiliser les faits des hôtes gérés comme variables sur un modèle.

**NOTE**

Pour rappel, les faits associés à un hôte géré peuvent être obtenus à l'aide de la commande **ansible system\_hostname -i inventory\_file -m setup**.

L'exemple suivant illustre la création d'un modèle avec des variables en utilisant deux des faits récupérés par Ansible auprès des hôtes gérés : `ansible_facts.hostname` et `ansible_facts.date_time.date`. Lors de l'exécution du playbook associé, ces deux faits sont remplacés par leurs valeurs dans l'hôte géré en cours de configuration.

**NOTE**

Aucune extension de fichier spécifique n'est nécessaire pour un fichier qui contient un modèle Jinja2 (.j2, par exemple). Cependant, fournir une telle extension de fichier peut vous aider à vous rappeler plus facilement qu'il s'agit d'un fichier modèle.

```
Welcome to {{ ansible_facts.hostname }}.
Today's date is: {{ ansible_facts.date_time.date }}.
```

## DÉPLOIEMENT DE MODÈLES JINJA2

Les modèles Jinja2 constituent un puissant outil pour personnaliser des fichiers de configuration à déployer sur les hôtes gérés. Dès que le modèle Jinja2 a été créé pour un fichier de configuration, il peut être déployé sur les hôtes gérés à l'aide du module `template`, lequel prend en charge le transfert d'un fichier local du nœud de contrôle vers les hôtes gérés.

Pour utiliser le module `template`, faites appel à la syntaxe suivante. La valeur associée à la clé `src` indique le modèle Jinja2 source, tandis que celle associée à la clé `dest` désigne le fichier à créer sur les hôtes de destination.

```
tasks:
  - name: template render
    template:
      src: /tmp/j2-template.j2
      dest: /tmp/dest-config-file.txt
```

**NOTE**

Le module `template` vous permet également de spécifier le propriétaire (l'utilisateur qui possède le fichier), le groupe, les autorisations et le contexte SELinux du fichier déployé, tout comme le module `file`. Il peut aussi prendre une option `validate` pour exécuter une commande arbitraire (telle que `visudo -c`) afin de vérifier l'exactitude de la syntaxe d'un fichier avant de le copier à l'endroit souhaité.

Pour obtenir plus de détails, consultez **ansible-doc template**.

## GESTION DE FICHIERS TRANSFORMÉS EN MODÈLES

Pour éviter que les administrateurs système modifient les fichiers déployés par Ansible, il est recommandé d'inclure un commentaire en haut du modèle pour indiquer que le fichier ne doit pas être modifié manuellement.

Une façon de procéder consiste à utiliser la chaîne « Ansible managed » définie dans la directive `ansible_managed`. Ce n'est pas une variable normale mais elle peut être utilisée comme celle d'un modèle. La directive `ansible_managed` est définie dans le fichier `ansible.cfg` :

```
ansible_managed = Ansible managed
```

Utilisez la syntaxe suivante pour inclure la chaîne `ansible_managed` dans un modèle Jinja2 :

```
{{ ansible_managed }}
```

## STRUCTURES DE CONTRÔLE

Vous pouvez utiliser les structures de contrôle Jinja2 dans les fichiers de modèle pour réduire les saisies répétitives, pour insérer les entrées de chaque hôte de manière dynamique dans un play ou pour insérer du texte de manière conditionnelle dans un fichier.

### Utilisation de boucles

Jinja2 utilise l'instruction `for` pour fournir la fonctionnalité de bouclage. Dans l'exemple suivant, la variable `user` est remplacée par toutes les valeurs incluses dans la variable `users`, une valeur par ligne.

```
{% for user in users %}
    {{ user }}
{% endfor %}
```

L'exemple de modèle suivant utilise une instruction `for` pour parcourir toutes les valeurs de la variable `users`, en remplaçant `myuser` par chaque valeur, sauf lorsque la valeur est `root`.

```
{# for statement #}
{% for myuser in users if not myuser == "root" %}
User number {{loop.index}} - {{ myuser }}
{% endfor %}
```

La variable `loop.index` s'étend jusqu'au numéro d'index atteint par la boucle active. Sa valeur est égale à 1 lors de la première exécution de la boucle et augmente d'une unité à chaque itération.

Autre exemple : ce modèle utilise également une instruction `for` et part du principe qu'une variable `myhosts` a été définie dans le fichier d'inventaire en cours d'utilisation. Cette variable contiendra une liste d'hôtes à gérer. Avec l'instruction `for` suivante, tous les hôtes du groupe `myhosts` de l'inventaire seront listés dans le fichier.

```
{% for myhost in groups['myhosts'] %}
{{ myhost }}
{% endfor %}
```

Si vous souhaitez un autre exemple, plus pratique, vous pouvez utiliser cela pour générer un fichier **/etc/hosts** à partir de faits d'hôte dynamiquement. Supposons que vous ayez le playbook suivant :

```
- 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
```

Le modèle **templates/hosts.j2** à 3 lignes suivant construit le fichier à partir de tous les hôtes du groupe **all**. (La ligne médiane est extrêmement longue dans le modèle en raison de la longueur des noms de variables.) Il effectue une itération sur chaque hôte du groupe pour obtenir trois faits pour le fichier **/etc/hosts**.

```
{% for host in groups['all'] %}
{{ hostvars[host]['ansible_facts']['default_ipv4']['address'] }} {{ hostvars[host]
['ansible_facts']['fqdn'] }} {{ hostvars[host]['ansible_facts']['hostname'] }}
{% endfor %}
```

## Utilisation de conditions

Jinja2 utilise l'instruction **if** pour fournir un contrôle conditionnel. Cela vous permet de mettre une ligne dans un fichier déployé si certaines conditions sont remplies.

Dans l'exemple suivant, la valeur de **result** variable est placée dans le fichier déployé uniquement si la valeur de **finished** variable est **Vrai** .

```
{% if finished %}
{{ result }}
{% endif %}
```



### IMPORTANT

Vous pouvez utiliser des boucles et des conditions Jinja2 dans les modèles Ansible, mais pas dans les playbooks Ansible.

## FILTRES DE VARIABLE

Jinja2 fournit des filtres qui modifient le format en sortie pour les expressions de modèle (au format JSON, par exemple). Des filtres sont disponibles pour des langages tels que YAML et JSON. Le filtre **to\_json** formate le résultat de l'expression à l'aide de JSON, tandis que le filtre **to\_yaml** formate le résultat de l'expression au moyen de YAML.

```
{{ output | to_json }}
{{ output | to_yaml }}
```

D'autres filtres, tels que **to\_nice\_json** et **to\_nice\_yaml**, convertissent le résultat de l'expression dans un format JSON ou YAML lisible.

```
{{ output | to_nice_json }}
{{ output | to_nice_yaml }}
```

Les filtres **from\_json** et **from\_yaml** attendent tous deux des chaînes au format JSON ou YAML pour les analyser.

```
{{ output | from_json }}
{{ output | from_yaml }}
```

Les expressions utilisées avec des clauses **when** dans les playbooks Ansible sont des expressions Jinja2. Les filtres Ansible intégrés utilisés pour tester les valeurs en retour sont notamment **failed**, **changed**, **succeeded** et **skipped**. La tâche suivante montre comment utiliser des filtres à l'intérieur d'expressions conditionnelles.

```
tasks:
  ...output omitted...
  - debug: msg="the execution was aborted"
    when: returnvalue | failed
```

**Figure 6.0: Déploiement de fichiers personnalisés avec des modèles Jinja2**



## RÉFÉRENCES

**template : Crée un fichier de modèle sur un serveur distant – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/modules/template\\_module.html](https://docs.ansible.com/ansible/2.7/modules/template_module.html)

**Variables – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks\\_variables.html](https://docs.ansible.com/ansible/2.7/user_guide/playbooks_variables.html)

**Filtres – Documentation Ansible**

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

## ► EXERCICE GUIDÉ

# DÉPLOIEMENT DE FICHIERS PERSONNALISÉS AVEC DES MODÈLES JINJA2

Dans cet exercice, vous allez créer un fichier modèle simple que votre playbook utilisera pour installer un fichier Message du jour personnalisé sur chaque hôte géré.

## RÉSULTATS

Vous devez pouvoir :

- Générer un fichier de modèle.
- Utiliser le fichier de modèle dans un playbook.

Connectez-vous à **workstation** en tant qu'utilisateur **student** avec le mot de passe **student**.

Sur **workstation**, exécutez le script **lab file-template setup**. Ce script s'assure qu'Ansible est installé sur **workstation**, crée le répertoire **/home/student/file-template** et télécharge le fichier **ansible.cfg** vers ce répertoire.

```
[student@workstation ~]$ lab file-template setup
```



### NOTE

Tous les fichiers utilisés lors de cet exercice sont disponibles pour référence sur **workstation**, dans le répertoire **/home/student/file-template/files**.

- 1. Sur **workstation**, accédez au répertoire de travail **/home/student/file-template**.

```
[student@workstation ~]$ cd ~/file-template
[student@workstation file-template]$
```

- 2. Créez le fichier **inventory** dans le répertoire de travail actuel. Ce fichier configure deux groupes : **webservers** et **workstations**. Placez le système **servera.lab.example.com** dans le groupe **webservers** et le système **workstation.lab.example.com** dans le groupe **workstations**.

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

```
[workstations]
workstation.lab.example.com
```

- 3. Créez un modèle pour le message du jour (Message of the Day) et incluez-le dans le fichier **motd.j2**, dans le répertoire de travail actuel. Entrez les variables suivantes dans le modèle :

- `ansible_hostname`, pour récupérer le nom d'hôte de l'hôte géré.
- `ansible_date_time.date`, pour la date de l'hôte géré.
- `system_owner`, pour l'e-mail du propriétaire du système. Cette variable doit obligatoirement être définie, avec une valeur appropriée, dans la section **vars** du modèle de playbook.

```
This is the system {{ ansible_hostname }}.  
Today's date is: {{ ansible_date_time.date }}.  
Only use this system with permission.  
You can ask {{ system_owner }} for access.
```

- 4. Créez un fichier de playbook appelé **motd.yml** dans le répertoire de travail actuel. Définissez la variable `system_owner` dans la section **vars** et insérez une tâche pour le module `template` qui associe le modèle Jinja2 **motd.j2** au fichier distant `/etc/motd` sur les hôtes gérés. Définissez le propriétaire et le groupe sur `root`, et le mode sur `0644`.

```
---  
- hosts: all  
  remote_user: devops  
  become: true  
  vars:  
    system_owner: clyde@example.com  
  tasks:  
    - template:  
        src: motd.j2  
        dest: /etc/motd  
        owner: root  
        group: root  
        mode: 0644
```

- 5. Avant d'exécuter le playbook, utilisez la commande **ansible-playbook --syntax-check** pour vérifier la syntaxe. En cas d'erreurs, corrigez-les avant de passer à l'étape suivante. Le résultat doit ressembler à ce qui suit :

```
[student@workstation file-template]$ ansible-playbook --syntax-check motd.yml  
playbook: motd.yml
```

- 6. Exécutez le playbook inclus dans le fichier **motd.yml**.

```
[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
```

- 7. Connectez-vous à `servera.lab.example.com` en tant qu'utilisateur `devops` pour vérifier que le message du jour (MOTD) est affiché lorsque vous êtes connecté. Déconnectez-vous lorsque vous avez terminé.

```
[student@workstation file-template]$ ssh devops@servera.lab.example.com
This is the system servera.
Today's date is: 2017-07-21.
Only use this system with permission.
You can ask clyde@example.com for access.
[devops@servera ~]# exit
Connection to servera.lab.example.com closed.
```

## Nettoyage

Exécutez la commande `lab file-template cleanup` pour procéder au nettoyage, une fois l'exercice terminé.

```
[student@workstation ~]$ lab file-template cleanup
```

## ► OPEN LAB

# DÉPLOIEMENT DE FICHIERS SUR DES HÔTES GÉRÉS

## LISTE DE CONTRÔLE DES PERFORMANCES

Dans cet atelier, vous allez exécuter un playbook qui crée un fichier personnalisé sur vos hôtes gérés à l'aide d'un modèle Jinja2.

## RÉSULTATS

Vous devez pouvoir :

- Générer un fichier de modèle.
- Utiliser le fichier de modèle dans un playbook.

Connectez-vous à `workstation` en tant qu'utilisateur `student` avec le mot de passe `student`.

Sur `workstation`, exécutez le script `lab file-review setup`. Il s'assure qu'Ansible est installé sur `workstation`, crée le répertoire `/home/student/file-review` et télécharge le fichier `ansible.cfg` vers ce répertoire. Il télécharge également les fichiers `motd.yml`, `motd.j2` et `inventory` dans le répertoire `/home/student/file-review/files`.

```
[student@workstation ~]$ lab file-review setup
```



### NOTE

Tous les fichiers utilisés dans cet exercice sont disponibles sur `workstation`, dans le répertoire `/home/student/file-review/files`.

1. Créez un fichier d'inventaire nommé `inventory` dans le répertoire `/home/student/file-review`. Ce fichier d'inventaire définit le groupe `servers` auquel est associé l'hôte géré `serverb.lab.example.com`.
2. Identifiez les faits sur `serverb.lab.example.com`, lesquels affichent l'état de la mémoire système.
3. Créez un modèle pour le message du jour (Message of the Day), nommé `motd.j2`, dans le répertoire de travail actuel. Quand l'utilisateur `devops` se connecte à `serverb.lab.example.com`, un message doit s'afficher pour indiquer la mémoire totale et la mémoire actuellement disponible sur le système. Utilisez les faits `ansible_memtotal_mb` et `ansible_memfree_mb` pour fournir les informations de la mémoire pour le message.
4. Créez un fichier de playbook appelé `motd.yml` dans le répertoire actif nommé. À l'aide du module `template`, configurez le fichier de modèle Jinja2 `motd.j2` créé précédemment, pour qu'il soit associé au fichier `/etc/motd` sur les hôtes gérés. Ce fichier est associé à l'utilisateur `root` comme propriétaire et groupe, et ses autorisations sont 0644. Configurez

le playbook de sorte qu'il fasse appel à l'utilisateur devops et définisse le paramètre become sur **true**.

5. Exécutez le playbook inclus dans le fichier **motd.yml**.
6. Vérifiez que le playbook inclus dans le fichier **motd.yml** a été exécuté correctement.

## Évaluation

À partir de workstation, exécutez le script **lab file-review grade** pour confirmer que l'exercice est réussi.

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

## Nettoyage

Sur workstation, exécutez le script **lab file-review cleanup** pour effacer après l'atelier.

```
[student@workstation ~]$ lab file-review cleanup
```

## ► SOLUTION

# DÉPLOIEMENT DE FICHIERS SUR DES HÔTES GÉRÉS

## LISTE DE CONTRÔLE DES PERFORMANCES

Dans cet atelier, vous allez exécuter un playbook qui crée un fichier personnalisé sur vos hôtes gérés à l'aide d'un modèle Jinja2.

## RÉSULTATS

Vous devez pouvoir :

- Générer un fichier de modèle.
- Utiliser le fichier de modèle dans un playbook.

Connectez-vous à **workstation** en tant qu'utilisateur **student** avec le mot de passe **student**.

Sur **workstation**, exécutez le script **lab file-review setup**. Il s'assure qu'Ansible est installé sur **workstation**, crée le répertoire **/home/student/file-review** et télécharge le fichier **ansible.cfg** vers ce répertoire. Il télécharge également les fichiers **motd.yml**, **motd.j2** et **inventory** dans le répertoire **/home/student/file-review/files**.

```
[student@workstation ~]$ lab file-review setup
```



### NOTE

Tous les fichiers utilisés dans cet exercice sont disponibles sur **workstation**, dans le répertoire **/home/student/file-review/files**.

1. Créez un fichier d'inventaire nommé **inventory** dans le répertoire **/home/student/file-review**. Ce fichier d'inventaire définit le groupe **servers** auquel est associé l'hôte géré **serverb.lab.example.com**.
  - 1.1. Sur **workstation**, accédez au répertoire **/home/student/file-review**.

```
[student@workstation ~]$ cd ~/file-review/
```

- 1.2. Créez le fichier **inventory** dans le répertoire actif. Ce fichier configure un seul groupe, appelé **servers**. Placez le système **serverb.lab.example.com** dans le groupe **servers**.

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

2. Identifiez les faits sur `serverb.lab.example.com`, lesquels affichent l'état de la mémoire système.
- 2.1. Utilisez le module `setup` pour obtenir la liste de tous les faits de l'hôte géré `serverb.lab.example.com`. Les faits `ansible_memfree_mb` et `ansible_memtotal_mb` fournissent des informations sur la mémoire libre et la mémoire totale de l'hôte géré.

```
[student@workstation file-review]$ ansible serverb.lab.example.com -m setup
serverb.lab.example.com | SUCCESS => {
    "ansible_facts": {
...output omitted...
    "ansible_memfree_mb": 157,
...output omitted...
    "ansible_memtotal_mb": 488,
...output omitted...
},
    "changed": false
}
```

3. Créez un modèle pour le message du jour (Message of the Day), nommé **motd.j2**, dans le répertoire de travail actuel. Quand l'utilisateur `devops` se connecte à `serverb.lab.example.com`, un message doit s'afficher pour indiquer la mémoire totale et la mémoire actuellement disponible sur le système. Utilisez les faits `ansible_memtotal_mb` et `ansible_memfree_mb` pour fournir les informations de la mémoire pour le message.
- 3.1. Créez un fichier nommé **motd.j2** dans le répertoire actif. Utilisez les variables de faits `ansible_memfree_mb` et `ansible_memtotal_mb` pour créer un message du jour.

```
[student@workstation file-review]$ cat motd.j2
This system's total memory is: {{ ansible_memtotal_mb }} MBs.
The current free memory is: {{ ansible_memfree_mb }} MBs.
```

4. Créez un fichier de playbook appelé **motd.yml** dans le répertoire actif nommé. À l'aide du module `template`, configurez le fichier de modèle Jinja2 **motd.j2** créé précédemment, pour qu'il soit associé au fichier `/etc/motd` sur les hôtes gérés. Ce fichier est associé à l'utilisateur `root` comme propriétaire et groupe, et ses autorisations sont 0644. Configurez le playbook de sorte qu'il fasse appel à l'utilisateur `devops` et définisse le paramètre `become` sur **true**.
- 4.1. Créez un fichier de playbook appelé **motd.yml** dans le répertoire actif nommé. À l'aide du module `template`, configurez le fichier de modèle Jinja2 **motd.j2** précédemment créé en tant que valeur pour le paramètre `src`, et `/etc/motd` en tant que valeur pour le paramètre `dest`. Configurez les paramètres `owner` et `group` sur `root`, et le paramètre `mode` sur **0644**. Utilisez l'utilisateur `devops` pour le paramètre `remote_user` et configurez le paramètre `become` sur **true**.

```
[student@workstation file-review]$ cat motd.yml
---
- hosts: all
  remote_user: devops
  become: true
  tasks:
```

```
- template:
  src: motd.j2
  dest: /etc/motd
  owner: root
  group: root
  mode: 0644
```

**5.** Exécutez le playbook inclus dans le fichier **motd.yml**.

- 5.1. Avant d'exécuter le playbook, utilisez la commande **ansible-playbook --syntax-check** pour vérifier sa syntaxe. En cas d'erreurs, corrigez-les avant de passer à l'étape suivante. Le résultat doit ressembler à ce qui suit :

```
[student@workstation file-review]$ ansible-playbook --syntax-check motd.yml
playbook: motd.yml
```

5.2. Exécutez le playbook inclus dans le fichier **motd.yml**.

```
[student@workstation file-review]$ ansible-playbook motd.yml
PLAY [all] ****
TASK [Gathering Facts] ****
ok: [serverb.lab.example.com]

TASK [template] ****
changed: [serverb.lab.example.com]

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

**6.** Vérifiez que le playbook inclus dans le fichier **motd.yml** a été exécuté correctement.

- 6.1. Connectez-vous à `serverb.lab.example.com` en tant qu'utilisateur `devops`, et vérifiez que le message du jour (MOTD) est affiché lors de la connexion. Déconnectez-vous lorsque vous avez terminé.

```
[student@workstation file-review]$ ssh devops@serverb.lab.example.com
This system's total memory is: 488 MBs.
The current free memory is: 162 MBs.
[devops@serverb ~]$ logout
```

## Évaluation

À partir de `workstation`, exéutez le script **lab file-review grade** pour confirmer que l'exercice est réussi.

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

## Nettoyage

Sur `workstation`, exéutez le script **lab file-review cleanup** pour effacer après l'atelier.

```
[student@workstation ~]$ lab file-review cleanup
```

# RÉSUMÉ

---

Dans ce chapitre, vous avez appris les principes suivants :

- La bibliothèque de modules `Files` inclut des modules qui vous permettent d'accomplir la plupart des tâches liées à la gestion de fichiers, telles que la création, la copie, l'édition et la modification des autorisations et autres attributs des fichiers.
- Vous pouvez utiliser les modèles `Jinja2` pour construire dynamiquement des fichiers à déployer.
- Un modèle `Jinja2` se compose généralement de deux éléments : des variables et des expressions. Ces variables et expressions sont remplacées par des valeurs lors du rendu du modèle `Jinja2`.
- Les filtres `Jinja2` transforment des expressions de modèle d'un type de données vers un autre.



## CHAPITRE 7

# GESTION DE PROJETS VOLUMINEUX

### PROJET

Rédiger des playbooks optimisés pour des projets plus grands et plus complexes.

### OBJECTIFS

- Écrire des modèles d'hôte sophistiqués pour sélectionner efficacement les hôtes d'une commande play ou ad hoc.
- Décrire ce que sont les inventaires dynamiques, et installer et utiliser un script existant en tant que source d'inventaire dynamique Ansible.
- Régler le nombre de connexions simultanées ouvertes par Ansible sur les hôtes gérés et déterminer comment Ansible traite les groupes d'hôtes gérés via les tâches du play.
- Gérer des playbooks volumineux en important ou en incluant d'autres playbooks ou tâches à partir de fichiers externes, de manière inconditionnelle ou sur la base d'un test conditionnel.

### SECTIONS

- Sélection d'hôtes avec des modèles d'hôte (et exercice guidé)
- Gestion des inventaires dynamiques (et exercice guidé)
- Configuration du parallélisme (et exercice guidé)
- Inclusion et importation de fichiers (et exercice guidé)

### ATELIER

- Gestion de projets volumineux

# SÉLECTION D'HÔTES AVEC DES MODELES D'HÔTE

## OBJECTIFS

Au terme de cette section, les stagiaires doivent pouvoir écrire des modèles d'hôte sophistiqués pour sélectionner efficacement les hôtes d'un play ou d'une commande ad hoc.

## RÉFÉRENCEMENT DES HÔTES D'INVENTAIRE

Les *modèles d'hôte* servent à spécifier les hôtes à cibler à l'aide d'un play ou d'une commande ad hoc. Dans sa forme la plus simple, le nom d'un hôte géré ou d'un groupes d'hôtes dans l'inventaire est un modèle d'hôte qui spécifie cet hôte ou ce groupe d'hôtes.

Vous avez déjà utilisé les modèles d'hôte dans ce cours. Dans un play, la directive `hosts` spécifie les hôtes gérés sur lesquels exécuter le play. Pour une commande ad hoc, le modèle d'hôte est fourni en tant qu'argument de ligne de commande à la commande **ansible**.

Il est important de comprendre les modèles d'hôte. Il est généralement plus facile de contrôler les hôtes qu'un play cible en utilisant avec précaution des modèles d'hôte et en disposant de groupes d'inventaire appropriés au lieu de définir des conditions complexes sur les tâches du play.

L'exemple d'inventaire suivant est utilisé tout au long de cette section pour illustrer les modèles d'hôte.

```
[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

Pour montrer comment les modèles d'hôte sont résolus, vous allez exécuter un playbook Ansible, **playbook.yml**, en utilisant différents modèles d'hôte pour cibler différents sous-ensembles d'hôtes gérés à partir de cet exemple d'inventaire.

## Hôtes gérés

Le modèle d'hôte le plus élémentaire est le nom d'un seul hôte géré listé dans l'inventaire. Cela spécifie que l'hôte sera le seul de l'inventaire qui sera traité par la commande **ansible**.

Lorsque le playbook est lancé, la première tâche **Gathering Facts** doit être exécutée sur tous les hôtes gérés correspondant au modèle d'hôte. (Après la première tâche, il est possible qu'une tâche échoue sur un hôte géré, ce qui a pour effet de la supprimer du play.)

N'oubliez pas qu'une adresse IP peut être listée de façon explicite dans l'inventaire à la place d'un nom d'hôte. Si c'est le cas, elle peut être utilisée de la même façon comme modèle d'hôte. Si l'adresse IP ne se trouve pas dans l'inventaire, vous ne pouvez pas l'utiliser pour spécifier l'hôte, même si l'adresse IP se rapporte à ce nom d'hôte dans le DNS.

L'exemple suivant montre comment un modèle d'hôte peut être utilisé pour référencer une adresse IP contenue dans un inventaire.

```
[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...
```



### NOTE

La référence à des hôtes gérés par adresse IP dans l'inventaire peut poser un problème : il peut s'avérer difficile de se rappeler quelle adresse IP correspond à quel hôte pour vos plays et commandes ad hoc. Vous pouvez arriver à la conclusion que vous devez spécifier l'hôte par son adresse IP à des fins de connexion, toutefois, car l'hôte ne peut pas, pour une raison quelconque, avoir de vrai nom d'hôte DNS.

Il est possible de pointer un alias vers une adresse IP particulière de votre inventaire en définissant la variable d'hôte `ansible_host`. Par exemple, vous pouvez avoir dans votre inventaire un hôte nommé **dummy.example**, puis des connexions directes utilisant ce nom vers l'adresse IP 192.168.2.1 en créant un fichier `host_vars/dummy.example` contenant la variable d'hôte suivante :

```
ansible_host: 192.168.2.1
```

## Groupes

Vous avez également déjà utilisé des groupes d'hôte de l'inventaire en tant que modèles d'hôte. Quand un nom de groupe est utilisé en tant que modèle hôte, il spécifie qu'Ansible va agir sur les hôtes qui sont membres du groupe.

```
[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...
```

N'oubliez pas qu'il existe un groupe spécial appelé `all`, qui correspond à tous les hôtes gérés de l'inventaire.

```
[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]
```

Il existe également un groupe spécial appelé `ungrouped`, qui correspond à tous les hôtes gérés de l'inventaire qui ne sont membres d'aucun autre groupe :

```
[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]
```

## Caractères génériques

Pour faire la même chose, une méthode différente du modèle d'hôte `all` consiste à utiliser le caractère générique de l'astérisque (\*), qui correspond à n'importe quelle chaîne. Si le modèle d'hôte est juste un astérisque entre guillemets, tous les hôtes de l'inventaire seront pris en compte.

```
[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]
```



### IMPORTANT

Certains caractères utilisés dans des modèles d'hôte ont également une signification pour le shell. Cela peut représenter un problème lorsque des modèles d'hôte sont utilisés pour exécuter des commandes ad hoc depuis la ligne de commande à l'aide d'`ansible`. Dans ce cas, une pratique recommandée consiste à citer les modèles d'hôte utilisés sur la ligne de commande pour les protéger contre toute expansion du shell non voulue.

De la même façon, dans un playbook Ansible, vous pouvez avoir besoin de placer votre modèle d'hôte entre guillemets simples afin de vous assurer qu'il soit correctement analysé si vous utilisez des caractères génériques spéciaux ou des caractères de liste :

```
...
hosts: '!test1.example.com,development'
```

L'astérisque peut également être utilisé comme globalisation de fichiers pour correspondre à n'importe quel hôte géré ou groupe contenant une chaîne en particulier.

Par exemple, le modèle d'hôte générique suivant correspond à tous les noms d'inventaire qui se terminent par `.example.com` :

```
[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]
```

L'exemple suivant utilise un modèle d'hôte générique pour correspondre aux hôtes ou aux groupes d'hôtes qui commencent par **192.168.2.** :

```
[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]
```

L'exemple suivant utilise un modèle d'hôte générique pour correspondre aux hôtes ou aux groupes d'hôtes qui commencent par **datacenter**.

```
[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]
```

**IMPORTANT**

Les modèles d'hôte génériques correspondent à l'ensemble des noms d'inventaire, des hôtes et des groupes d'hôtes. Ils ne font pas de distinction entre les noms qui sont des noms DNS, des adresses IP ou des groupes. Cela peut entraîner des correspondances inattendues si vous oubliez ce fait.

Par exemple, compte tenu de l'exemple d'inventaire, comparez les résultats obtenus en spécifiant le modèle d'hôte **datacenter\*** de l'exemple précédent avec les résultats du modèle d'hôte **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]
```

## Listes

Il est possible de référencer plusieurs entrées d'un inventaire à l'aide de listes logiques. Une liste séparée par des virgules établit la correspondance avec tous les hôtes qui correspondent à un de ces modèles d'hôtes.

Si vous fournissez une liste séparée par des virgules d'hôtes gérés, tous ces hôtes gérés vont être ciblés :

```
[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]
```

Si vous fournissez une liste séparée par des virgules de groupes, tous les hôtes présents dans l'un de ces groupes vont être ciblés :

```
[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]
```

Vous pouvez également mélanger hôtes gérés, groupes d'hôtes et caractères génériques, comme illustré ci-dessous :

```
[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]
```



### NOTE

Il est possible d'utiliser les deux-points (:) pour remplacer la virgule. Toutefois, la virgule est la syntaxe préférée, en particulier lorsque l'on travaille avec des adresses IPv6 comme noms d'hôtes gérés. Vous verrez peut-être la syntaxe du point-virgule dans des exemples plus anciens.

Si un élément d'une liste commence par une esperluette (&), les hôtes doivent correspondre à cet élément pour correspondre au modèle d'hôte. Il fonctionne de la même façon qu'un AND logique.

Par exemple, en se basant sur notre exemple d'inventaire, le modèle d'hôte suivant correspond aux machines du groupe **lab** uniquement si elles se trouvent également dans le groupe **datacenter1**:

```
[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]
```

Vous pouvez également spécifier que les machines du groupe **datacenter1** ne correspondent que si elles se trouvent dans le groupe **lab** avec les modèles d'hôte **&lab**, **datacenter1** ou **datacenter1, &lab**.

Vous pouvez exclure les hôtes qui correspondent à un modèle provenant d'une liste en utilisant un point d'exclamation (!) devant le modèle d'hôte. Il fonctionne comme un NOT logique.

Cet exemple, en se basant sur notre inventaire de test, correspond à tous les hôtes définis dans le groupe **datacenter**, à l'exception de **test2.example.com**:

```
[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]
```

Le modèle '**!test2.example.com, datacenter**' aurait pu être utilisé dans l'exemple précédent pour obtenir le même effet.

L'exemple final montre l'utilisation d'un modèle d'hôte qui correspond à tous les hôtes de l'inventaire de test, à l'exception des hôtes gérés du groupe **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]



## RÉFÉRENCES

### **Utilisation de modèles – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/user\\_guide/intro\\_patterns.html](https://docs.ansible.com/ansible/2.7/user_guide/intro_patterns.html)

### **Utilisation d'un inventaire – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/user\\_guide/intro\\_inventory.html](https://docs.ansible.com/ansible/2.7/user_guide/intro_inventory.html)

## ► EXERCICE GUIDÉ

# SÉLECTION D'HÔTES AVEC DES MODÈLES D'HÔTE

Dans cet exercice, vous allez découvrir comment utiliser les modèles d'hôte pour spécifier les hôtes de l'inventaire pour les plays ou les commandes ad hoc. Plusieurs exemples d'inventaires vous seront fournis pour explorer les modèles d'hôte.

## RÉSULTATS

Vous devez pouvoir utiliser différents modèles d'hôte pour accéder à différents hôtes dans un inventaire.

Connectez-vous à `workstation` en tant qu'utilisateur `student` avec le mot de passe `student`. Exécutez la commande `lab projects-host setup`.

```
[student@workstation ~]$ lab projects-host setup
```

Le script de configuration confirme qu'Ansible est installé sur `workstation` et crée une structure de répertoire pour l'environnement de l'atelier.

- 1. Sur `workstation`, accédez au répertoire de travail de l'exercice, `/home/student/projects-host` et examinez-en le contenu.

```
[student@workstation ~]$ cd /home/student/projects-host  
[student@workstation projects-host]$
```

- 1.1. Listez le contenu de ce répertoire.

```
[student@workstation projects-host]$ ls  
ansible.cfg inventory1 inventory2 playbook.yml
```

- 1.2. Inspectez l'exemple de fichier d'inventaire, `inventory1`. Notez comment l'inventaire est organisé. Explorez pour voir quels hôtes se trouvent dans l'inventaire, quels domaines sont utilisés et quels groupes se trouvent dans cet inventaire.

```
[student@workstation projects-host]$ cat inventory1  
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. Inspectez l'exemple de fichier d'inventaire, **inventory2**. Notez comment l'inventaire est organisé. Explorez pour voir quels hôtes se trouvent dans l'inventaire, quels domaines sont utilisés et quels groupes se trouvent dans cet inventaire.

```
[student@workstation projects-host]$ cat inventory2
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. Enfin, inspectez le contenu du playbook, **playbook.yml**. Remarquez comment le playbook utilise le module debug pour afficher le nom de chaque hôte géré.

```
[student@workstation projects-host]$ cat playbook.yml
---
- name: Resolve host patterns
  hosts:
    tasks:
      - name: Display managed host name
        debug:
          msg: "{{ inventory_hostname }}"
```

- ▶ 2. À l'aide d'une commande ad hoc, déterminez si le serveur db1.example.com est présent dans le fichier d'inventaire **inventory1**.

```
[student@workstation projects-host]$ ansible db1.example.com -i inventory1 \
> --list-hosts
hosts (1):
db1.example.com
```

- ▶ 3. À l'aide d'une commande ad hoc, faites référence à une adresse IP contenue dans l'inventaire **inventory1** avec un modèle d'hôte.

```
[student@workstation projects-host]$ ansible 172.25.252.44 -i inventory1 \
> --list-hosts
hosts (1):
```

172.25.252.44

- ▶ 4. Avec une commande ad hoc, utilisez le groupe **all** pour lister tous les hôtes gérés du fichier d'inventaire **inventory1**.

```
[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. Avec une commande ad hoc, utilisez l'astérisque (\*) pour lister tous les hôtes qui se terminent par **.example.com** dans le fichier d'inventaire **inventory1**.

```
[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. Comme vous pouvez le voir dans la sortie de la commande suivante, il y a 14 hôtes dans le domaine **\*.example.com**. Modifiez le modèle d'hôte dans la commande ad hoc précédente afin que les hôtes du domaine **\*.lab.example.com** soient ignorés.

```
[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. En utilisant une commande ad hoc, sans accéder aux groupes du fichier d'inventaire **inventory1**, listez ces trois hôtes : `lb1.lab.example.com`, `s1.lab.example.com` et `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. Utilisez un modèle d'hôte générique dans une commande ad hoc pour lister les hôtes dont l'adresse IP commence par **172.25**. Adresse IP dans le fichier d'inventaire **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. Utilisez un modèle d'hôte dans une commande ad hoc pour lister tous les hôtes commençant par la lettre « **s** » dans le fichier d'inventaire **inventory1**.

```
[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
```

Notez les hôtes `file2.example.com` et `db2.example.com` dans la sortie de la commande précédente. Ils sont affichés dans la liste car ils sont tous deux membres d'un groupe nommé `stage`, qui commence également par la lettre « **s** ».

- ▶ 10. À l'aide d'une liste et de modèles d'hôtes génériques, listez tous les hôtes figurant dans l'inventaire **inventory1** du groupe `prod`, ou dont l'adresse IP commence par **172** ou encore dont le nom contient **lab**.

```
[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. Utilisez une commande ad hoc pour lister tous les hôtes qui appartiennent à la fois aux groupes db et london.

```
[student@workstation projects-host]$ ansible 'db,&london' -i inventory1 \
> --list-hosts
hosts (2):
  db2.example.com
  db3.example.com
```

- 12. Modifiez le modèle d'hôte fourni en tant que valeur du mot-clé **hosts** dans le playbook **playbook.yml** de façon à ce que tous les serveurs du groupe london soient ciblés. Exécutez le playbook à l'aide du fichier d'inventaire **inventory2**.

```
[student@workstation projects-host]$ cat playbook.yml
...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. Modifiez le modèle d'hôte fourni en tant que valeur du mot-clé **hosts** dans le playbook **playbook.yml** de façon à ce que tous les serveurs du groupe imbriqué europe soient ciblés. Exécutez le playbook à l'aide du fichier d'inventaire **inventory2**.

```
[student@workstation projects-host]$ cat playbook.yml
...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. Modifiez le modèle d'hôte fourni en tant que valeur du mot-clé **hosts** dans le playbook **playbook.yml** de façon à ce que tous les serveurs n'appartenant à aucun groupe soient ciblés. Exécutez le playbook à l'aide du fichier d'inventaire **inventory2**.

```
[student@workstation projects-host]$ cat playbook.yml
...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...
```

## Nettoyage

Sur workstation, exécutez le script **lab projects-host cleanup** pour effacer l'exercice.

```
[student@workstation ~]$ lab projects-host cleanup
```

L'exercice guidé est maintenant terminé.

# GESTION DES INVENTAIRES DYNAMIQUES

## OBJECTIFS

Au terme de cette section, les stagiaires doivent pouvoir décrire ce que sont les inventaires dynamiques, et installer et utiliser un script existant en tant que source d'inventaire dynamique Ansible.

## GÉNÉRATION DYNAMIQUE DES INVENTAIRES

Les fichiers d'inventaire statiques avec lesquels vous avez travaillé jusqu'à présent sont faciles à écrire et sont pratiques pour la gestion de petites infrastructures. Toutefois, lorsque vous travaillez avec un grand nombre de machines ou dans un environnement où les machines vont et viennent, il peut être difficile de maintenir les fichiers d'inventaire statiques à jour.

La plupart des grands environnements informatiques ont des systèmes qui gardent la trace des hôtes disponibles et de la façon dont ils sont organisés. Par exemple, un service d'annuaire externe peut être géré par un système de surveillance tel que Zabbix, ou sur des serveurs FreeIPA ou Active Directory. Les serveurs d'installation tels que Cobbler ou les services de gestion tels que Red Hat Satellite peuvent suivre les systèmes nus qui sont déployés. De la même manière, les services cloud tels qu'Amazon Web Services EC2 ou un déploiement OpenStack, ou les infrastructures de machines virtuelles basées sur VMware ou Red Hat Virtualization peuvent être des sources d'informations sur les instances et les machines virtuelles qui vont et viennent.

Ansible prend en charge les scripts d'*inventaire dynamique* qui récupèrent les informations actuelles à partir de ces types de sources à chaque exécution d'Ansible, autorisant ainsi la mise à jour de l'inventaire en temps réel. Ces scripts sont des programmes exécutables qui collectent des informations à partir d'une source externe et génèrent l'inventaire au format JSON.

Les scripts d'inventaire dynamique sont utilisés de la même manière que les fichiers de texte d'inventaire statiques. L'emplacement de l'inventaire est spécifié directement dans le fichier **ansible.cfg** actif, ou à l'aide de l'option **-i**. Si le fichier d'inventaire est exécutable, il est traité comme un programme d'inventaire dynamique et Ansible tente de l'exécuter afin de générer l'inventaire. Si le fichier n'est pas exécutable, il est traité comme un inventaire statique.



### NOTE

L'emplacement de l'inventaire peut être configuré dans le fichier de configuration **ansible.cfg** à l'aide du paramètre **inventory**. Par défaut, il est configuré sur **/etc/ansible/hosts**.

## CONTRIBUTION DE SCRIPTS

Un certain nombre de scripts d'inventaire dynamique existants ont été fournis au projet Ansible par la communauté Open Source. Ils ne sont pas inclus dans le paquetage **ansible** ni officiellement pris en charge par Red Hat. Ils sont disponibles sur le site Ansible GitHub à l'adresse <https://github.com/ansible/ansible/tree/devel/contrib/inventory>.

Certaines sources de données ou plateformes ciblées par les scripts d'inventaire dynamique fournis comprennent :

- Des plateformes de cloud privées, comme Red Hat OpenStack Platform.
- Des plateformes de cloud publiques, comme Rackspace Cloud, Amazon Web Services EC2 et Google Compute Engine.
- Des plateformes de virtualisation, telles que Red Hat Virtualization (oVirt) et VMware vSphere.
- Des solutions « Platform-as-a-Service » (PaaS), comme OpenShift ContainerPlatform.
- Des outils de gestion du cycle de vie, comme Foreman (avec Red Hat Satellite 6 ou version autonome) et Spacewalk (en amont de Red Hat Satellite 5).
- Des fournisseurs d'hébergement, comme Digital Ocean et Linode.

Chaque script peut avoir ses propres dépendances et exigences pour fonctionner. Les scripts fournis sont, pour la plupart, écrits en Python, mais ce n'est pas obligatoire pour les scripts d'inventaire dynamique.

## ÉCRITURE DE PROGRAMMES D'INVENTAIRE DYNAMIQUE

Si un script d'inventaire dynamique n'existe pas pour l'infrastructure ou le système de répertoire utilisé, il est possible d'écrire un programme d'inventaire dynamique personnalisé. Il peut être écrit dans n'importe quel langage de programmation, et il doit renvoyer des informations d'inventaire au format JSON lorsque des options appropriées lui sont transmises.

La commande **ansible-inventory** peut être un outil utile pour apprendre à créer des inventaires Ansible au format JSON. Vous pouvez utiliser la commande **ansible-inventory**, disponible depuis Ansible 2.4, pour afficher un fichier d'inventaire au format JSON.

Pour afficher le contenu du fichier d'inventaire au format JSON, exécutez la commande **ansible-inventory --list**. Vous pouvez utiliser l'option **-i** pour spécifier l'emplacement du fichier d'inventaire à traiter ou simplement utiliser l'inventaire par défaut défini par la configuration Ansible actuelle.

L'exemple suivant illustre l'utilisation de la commande **ansible-inventory** pour traiter un fichier d'inventaire de type INI et une sortie au format JSON.

```
[student@workstation projects-host]$ cat inventory
workstation1.lab.example.com

[webservers]
web1.lab.example.com
web2.lab.example.com

[databases]
db1.lab.example.com
db2.lab.example.com

[student@workstation projects-host]$ ansible-inventory -i inventory --list
{
    "_meta": {
        "hostvars": {
            "db1.lab.example.com": {},
            "db2.lab.example.com": {},
            "web1.lab.example.com": {},
            "web2.lab.example.com": {}
        }
    }
}
```

```

        "workstation1.lab.example.com": {}
    }
},
"all": {
    "children": [
        "databases",
        "ungrouped",
        "webservers"
    ]
},
"databases": {
    "hosts": [
        "db1.lab.example.com",
        "db2.lab.example.com"
    ]
},
"ungrouped": {
    "hosts": [
        "workstation1.lab.example.com"
    ]
},
"webservers": {
    "hosts": [
        "web1.lab.example.com",
        "web2.lab.example.com"
    ]
}
}
}

```

Si vous voulez écrire votre propre script d'inventaire dynamique, davantage d'informations détaillées sont disponibles sur Developing Dynamic Inventory Sources [[http://docs.ansible.com/ansible/dev\\_guide/developing\\_inventory.html](http://docs.ansible.com/ansible/dev_guide/developing_inventory.html)] dans le *Ansible Developer Guide*. Voici un bref aperçu.

Le script doit commencer par une ligne « shebang » appropriée (par exemple, **`#!/usr/bin/python`**) et doit être exécutable pour être lancé par Ansible.

Une fois l'option **`--list`** passée, le script doit générer un hachage/dictionnaire codé au format JSON de tous les hôtes et groupes de l'inventaire vers la sortie standard.

Dans sa forme la plus simple, un groupe peut être une liste d'hôtes gérés. Dans cet exemple de la sortie codée au format JSON d'un script d'inventaire, `webservers` est un groupe d'hôtes dont `web1.lab.example.com` et `web2.lab.example.com` sont des hôtes gérés dans le groupe. Le groupe d'hôtes `databases` inclut les hôtes `db1.lab.example.com` et `db2.lab.example.com` comme membres.

```

[student@workstation ~]$ ./inventoryscript --list
{
    "webservers" : [ "web1.lab.example.com", "web2.lab.example.com" ],
    "databases"   : [ "db1.lab.example.com", "db2.lab.example.com" ]
}

```

Sinon, la valeur de chaque groupe peut être un hachage/dictionnaire JSON contenant une liste de chaque hôte géré, de tous les groupes enfants et de toutes les variables de groupe pouvant être

définies. L'exemple suivant montre la sortie codée au format JSON pour un inventaire dynamique plus complexe. Le groupe **boston** contient deux groupes enfants (**backup** et **ipa**), trois hôtes gérés propres, et un ensemble de variables de groupe (**example\_host: false**).

```
{
  "webservers" : [
    "web1.demo.example.com",
    "web2.demo.example.com"
  ],
  "boston" : {
    "children" : [
      "backup",
      "ipa"
    ],
    "vars" : {
      "example_host" : false
    },
    "hosts" : [
      "server1.demo.example.com",
      "server2.demo.example.com",
      "server3.demo.example.com"
    ]
  },
  "backup" : [
    "server4.demo.example.com"
  ],
  "ipa" : [
    "server5.demo.example.com"
  ],
  "_meta" : {
    "hostvars" : {
      "server5.demo.example.com": {
        "ntpserver": "ntp.demo.example.com",
        "dnsserver": "dns.demo.example.com"
      }
    }
  }
}
```

Le script doit également prendre en charge l'option **--host managed-host**. Cette option peut imprimer un hachage/dictionnaire JSON composé de variables qui doivent être associées à cet hôte. Si ce n'est pas le cas, un hachage/dictionnaire JSON vide doit être imprimé.

```
[student@workstation ~]$ ./inventoryscript --host server5.demo.example.com
{
  "ntpserver" : "ntp.demo.example.com",
  "dnsserver" : "dns.demo.example.com"
}
```

**NOTE**

Lorsqu'il est appelé avec l'option **--host hostname**, le script doit imprimer un hachage/dictionnaire JSON des variables pour l'hôte spécifié (potentiellement, un hachage ou un dictionnaire JSON vide si aucune variable n'est fournie).

En option, si **--list** renvoie un élément de niveau supérieur appelé **\_meta**, il est possible de renvoyer toutes les variables d'hôte dans un appel de script, ce qui améliore les performances des scripts. Dans ce cas, les appels **--host** ne se font pas.

Pour plus d'informations, consultez Developing Dynamic Inventory Sources [[http://docs.ansible.com/ansible/developing\\_inventory.html](http://docs.ansible.com/ansible/developing_inventory.html)].

## GESTION D'INVENTAIRES MULTIPLES

Ansible prend en charge l'utilisation de plusieurs inventaires dans une même exécution. Si l'emplacement de l'inventaire est un répertoire (qu'il soit défini par l'option **-i**, la valeur du paramètre **inventory**, ou de toute autre façon), tous les fichiers d'inventaire inclus dans le répertoire, qu'ils soient statiques ou dynamiques, sont combinés pour déterminer l'inventaire. Les fichiers exécutables de ce répertoire sont utilisés pour récupérer des inventaires dynamiques, et les autres fichiers sont utilisés en tant qu'inventaires statiques.

Les fichiers d'inventaire ne doivent pas dépendre d'autres fichiers d'inventaire ou scripts pour être résolus. Par exemple, si un fichier d'inventaire statique spécifie qu'un groupe particulier doit être un enfant d'un autre groupe, il doit également disposer d'une entrée d'espace réservé pour ce groupe, même si tous les membres de ce groupe proviennent de l'inventaire dynamique. Prenons le groupe **cloud-east** dans l'exemple suivant :

```
[cloud-east]

[servers]
test.demo.example.com

[servers:children]
cloud-east
```

Ceci garantit que, quel que soit l'ordre dans lequel les fichiers d'inventaire sont analysés, ils sont tous cohérents entre eux.

**NOTE**

L'ordre dans lequel les fichiers d'inventaire sont analysés n'est pas spécifié par la documentation. Actuellement, lorsque plusieurs fichiers d'inventaire existent, ils semblent être analysés par ordre alphabétique. Si une source d'inventaire dépend des informations d'une autre pour être logique, l'ordre dans lequel les fichiers sont chargés a une incidence sur la réussite de l'opération ou la génération d'une erreur. Par conséquent, il est important de s'assurer que tous les fichiers sont auto-cohérents pour éviter les erreurs inattendues.

Ansible ignore les fichiers d'un répertoire d'inventaire si leur nom finit par certains suffixes. Cela peut être contrôlé avec la directive **inventory\_ignore\_extensions** dans le fichier de

configuration Ansible en cours d'utilisation. Des informations supplémentaires sont disponibles dans la documentation Ansible.



## RÉFÉRENCES

### **Utilisation d'un inventaire dynamique : Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/user\\_guide/intro\\_dynamic\\_inventory.html](https://docs.ansible.com/ansible/2.7/user_guide/intro_dynamic_inventory.html)

### **Développement d'inventaire dynamique : Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/dev\\_guide/developing\\_inventory.html](https://docs.ansible.com/ansible/2.7/dev_guide/developing_inventory.html)

## ► EXERCICE GUIDÉ

# GESTION DES INVENTAIRES DYNAMIQUES

Dans cet exercice, vous allez installer des scripts personnalisés qui génèrent dynamiquement une liste d'hôtes d'inventaire.

## RÉSULTATS

Vous devez pouvoir installer et utiliser des scripts d'inventaire dynamique existants.

Connectez-vous à **workstation** en tant qu'utilisateur **student** avec le mot de passe **student**.

Sur **workstation**, exécutez le script **lab projects-inventory setup**. Il vérifie si Ansible est installé sur **workstation** et crée également un répertoire de travail pour cet exercice.

```
[student@workstation ~]$ lab projects-inventory setup
```

- 1. Sur **workstation**, accédez au répertoire de travail pour l'exercice, **/home/student/projects-inventory**.

```
[student@workstation ~]$ cd /home/student/projects-inventory
```

- 2. Affichez le contenu du fichier de configuration Ansible **ansible.cfg** dans le répertoire de travail. Le fichier de configuration définit l'emplacement de l'inventaire sur **inventory**.

```
[defaults]
inventory = inventory
```

- 3. Créez le répertoire **/home/student/projects-inventory/inventory**.

```
[student@workstation projects-inventory]$ mkdir inventory
```

- 4. À partir de <http://materials.example.com/labs/projects-inventory/>, téléchargez les fichiers **inventorya.py**, **inventoryw.py** et **hosts** dans votre répertoire **/home/student/projects-inventory/inventory**. Les deux fichiers se terminant

par **.py** sont des scripts qui génèrent des inventaires dynamiques, et le troisième fichier est un inventaire statique.

- Le script **inventorya.py** fournit le groupe **webservers**, qui inclut l'hôte **servera.lab.example.com**.
- Le script **inventoryw.py** fournit l'hôte **workstation.lab.example.com**.
- L'inventaire statique **hosts** définit le groupe **servers**, groupe parent du groupe **webservers**.

```
[student@workstation projects-inventory]$ wget http://materials.example.com/labs/projects-inventory/inventorya.py -O inventory/inventorya.py  
[student@workstation projects-inventory]$ wget http://materials.example.com/labs/projects-inventory/inventoryw.py -O inventory/inventoryw.py  
[student@workstation projects-inventory]$ wget http://materials.example.com/labs/projects-inventory/hosts -O inventory/hosts
```

- 5. En utilisant la commande **ansible** avec le script **inventorya.py** comme inventaire, liste les hôtes gérés associés au groupe **webservers**. Cela devrait déclencher une erreur relative aux autorisations de **inventorya.py**.

```
[student@workstation projects-inventory]$ ansible -i inventory/inventorya.py webservers --list-hosts  
[WARNING]: * Failed to parse /home/student/projects-inventory/inventory/inventorya.py with script plugin: problem running /home/student/projects-inventory/inventory/inventorya.py --list ([Errno 13] Permission denied)  
  
[WARNING]: * Failed to parse /home/student/projects-inventory/inventory/inventorya.py with ini plugin: /home/student/projects-inventory/inventory/inventorya.py:3: Expected key=value host variable assignment, got: subprocess  
  
[WARNING]: Unable to parse /home/student/projects-inventory/inventory/inventorya.py as an inventory source  
  
[WARNING]: No inventory was parsed, only implicit localhost is available  
  
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'  
  
[WARNING]: Could not match supplied host pattern, ignoring: webservers  
  
hosts (0):
```

- 6. Vérifiez les autorisations actuelles du script **inventorya.py** et remplacez-les par 755.

```
[student@workstation projects-inventory]$ ls -la inventory/inventorya.py  
-rw-rw-r-- 1 student student 639 Apr 29 14:20 inventory/inventorya.py  
[student@workstation projects-inventory]$ chmod 755 inventory/inventorya.py
```

- 7. Remplacez les autorisations du script **inventoryw.py** par 755.

```
[student@workstation projects-inventory]$ chmod 755 inventory/inventoryw.py
```

- 8. Contrôlez le résultat du script **inventorya.py** à l'aide du paramètre **--list**. Les hôtes associés au groupe **webservers** s'affichent.

```
[student@workstation projects-inventory]$ inventory/inventorya.py --list
{"webservers": {"hosts": ["servera.lab.example.com"], "vars": {}}}
```

- 9. Contrôlez le résultat du script **inventoryw.py** à l'aide du paramètre **--list**. L'hôte **workstation.lab.example.com** s'affiche.

```
[student@workstation projects-inventory]$ inventory/inventoryw.py --list
{"all": {"hosts": ["workstation.lab.example.com"], "vars": {}}}
```

- 10. Consultez la définition du groupe **servers** dans le fichier **/home/student/projects-inventory/inventory/hosts**. Le groupe **webservers** défini dans l'inventaire dynamique est configuré en tant qu'enfant du groupe **servers**.

```
[student@workstation projects-inventory]$ cat inventory/hosts
[servers:children]
webservers
```

- 11. Exécutez la commande suivante pour vérifier la liste des hôtes dans le groupe **webservers**. Elle génère une erreur liée au groupe **webservers** non défini.

```
[student@workstation projects-inventory]$ ansible webservers --list-hosts
[WARNING]: * Failed to parse /home/student/projects-inventory/inventory/hosts
with yaml plugin: Syntax Error while loading YAML.  found unexpected ':'
The error appears to have been in '/home/student/projects-inventory/inventory/hosts':
line 1, column 9, but may be elsewhere in the file depending on the exact syntax
problem.  The offending line appears to be:  [servers:children]           ^ here

[WARNING]: * Failed to parse /home/student/projects-inventory/inventory/hosts
with ini plugin: /home/student/projects-inventory/inventory/hosts:2: Section
[servers:children] includes undefined group: webservers

[WARNING]: Unable to parse /home/student/projects-inventory/inventory/hosts as an
inventory source

hosts (1):
  servera.lab.example.com
```

- 12. Pour éviter ce problème, l'inventaire statique doit avoir une entrée d'espace réservé qui définit un groupe d'hôtes **webservers** vide. Il est important que l'inventaire

statique définisse tout groupe d'hôtes auquel il fait référence, car il risque de disparaître dynamiquement de la source externe et provoquerait cette erreur.

Modifiez le fichier **/home/student/projects-inventory/inventory/hosts** afin qu'il contienne le contenu suivant :

```
[webservers]
```

```
[servers:children]
webservers
```



#### IMPORTANT

Si le script d'inventaire dynamique qui fournit le groupe d'hôtes est nommé de sorte qu'il est trié avant l'inventaire statique dans lequel il est référencé, il se peut que cette erreur n'apparaisse pas. Toutefois, si le groupe d'hôtes disparaît de l'inventaire dynamique, et que vous ne le faites pas, l'inventaire statique référencera un groupe d'hôtes manquant et l'erreur interrompra l'analyse de l'inventaire.

- 13. Exécutez à nouveau la commande suivante pour vérifier la liste des hôtes dans le groupe **webservers**. Elle doit fonctionner sans erreur.

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

## Nettoyage

Sur workstation, exécutez le script **lab projects-inventory cleanup** pour effacer l'exercice.

```
[student@workstation ~]$ lab projects-inventory cleanup
```

L'exercice guidé est maintenant terminé.

# CONFIGURATION D'UN PARALLÉLISME

## OBJECTIFS

Au terme de cette section, les stagiaires doivent pouvoir régler le nombre de connexions simultanées ouvertes par Ansible sur les hôtes gérés et déterminer comment Ansible traite les groupes d'hôtes gérés via les tâches du play.

## CONFIGURER LE PARALLÉLISME DANS ANSIBLE À L'AIDE DE FORKS

Lorsqu'Ansible traite un playbook, il lance chaque play dans l'ordre. Après avoir déterminé la liste des hôtes pour le play, il exécute chaque tâche dans l'ordre. Normalement, tous les hôtes doivent mener à bien une tâche avant qu'un hôte, quel qu'il soit, ne commence la tâche suivante du play.

En théorie, Ansible pourrait se connecter simultanément à tous les hôtes du play pour chaque tâche. Cela fonctionne très bien pour les petites listes d'hôtes. Mais si le play cible des centaines d'hôtes, le nœud de contrôle risque d'être lourdement chargé.

Le nombre maximal de connexions simultanées établies par Ansible est contrôlé par le paramètre **forks** dans le fichier de configuration Ansible. Il est mis à **5** par défaut.

```
[student@demo ~]$ grep forks ansible.cfg
forks      = 5
```

Par exemple, supposons qu'un nœud de contrôle Ansible soit configuré avec la valeur par défaut de cinq forks et que le play comporte dix hôtes gérés. Ansible exécutera la première tâche du play sur les cinq premiers hôtes gérés, suivie d'un second tour d'exécution de la première tâche sur les cinq autres hôtes gérés. Une fois que la première tâche a été exécutée sur tous les hôtes gérés, elle exécute ensuite la tâche suivante sur tous les hôtes gérés, dans des groupes de cinq hôtes à la fois. Cela se fera tour à tour avec chaque tâche jusqu'à la fin du play.

La valeur par défaut pour **forks** est définie de façon très conservatrice. Si votre nœud de contrôle gère des hôtes Linux, la plupart des tâches s'exécutent sur les hôtes gérés et le nœud de contrôle est moins chargé. Dans ce cas, vous pouvez généralement définir **forks** sur une valeur beaucoup plus élevée, peut-être plus proche de 100, et voir des améliorations de performances.

Si vos playbooks exécutent beaucoup de code sur le nœud de contrôle, vous devez augmenter judicieusement la limite de fork. Cela est vrai si vous utilisez Ansible pour gérer les routeurs et les commutateurs réseau, car la plupart de ces modules s'exécutent sur le nœud de contrôle, et non sur le périphérique réseau. En raison de la charge plus élevée que cela impose au nœud de contrôle, sa capacité à prendre en charge l'augmentation du nombre de forks sera considérablement inférieure à celle d'un nœud de contrôle ne gérant que des hôtes Linux.

Vous pouvez également remplacer le paramètre pour **forks** dans le fichier de configuration Ansible à partir de la ligne de commande. Les deux commandes **ansible** et **ansible-playbook** offrent les options **-f** ou **--forks** pour spécifier le nombre de forks à utiliser.

## GESTION DES MISES À JOUR PROGRESSIVES

Normalement, lorsque Ansible exécute un play, il s'assure que tous les hôtes gérés ont terminé chaque tâche avant de démarrer un hôte sur la tâche suivante. Une fois que tous les hôtes gérés ont terminé toutes les tâches, tous les gestionnaires notifiés sont exécutés.

Cela peut toutefois entraîner des effets secondaires indésirables. Par exemple, si un play met à jour un cluster de serveurs Web à charge équilibrée, il peut être nécessaire de mettre chaque serveur Web hors service pendant la mise à jour. Si tous les serveurs sont mis à jour dans le même play, ils pourraient tous être hors service en même temps.

Une façon d'éviter cela consiste à utiliser le mot-clé `serial` pour exécuter les hôtes à travers le play par lots. Chaque lot d'hôtes sera exécuté dans l'ensemble du play avant le démarrage du lot suivant.

Dans l'exemple ci-dessous, Ansible exécute le play sur deux hôtes gérés à la fois, jusqu'à ce que tous les hôtes gérés aient été mis à jour. Ansible commence par exécuter les tâches du jeu sur les deux premiers hôtes gérés. Si l'un des hôtes, ou les deux, ont notifié le gestionnaire, Ansible l'exécute en fonction des besoins de ces deux hôtes. Lorsque l'exécution du play est terminée sur ces deux hôtes gérés, Ansible répète le processus sur les deux hôtes gérés suivants. Ansible continue à exécuter le play de cette manière jusqu'à ce que tous les hôtes gérés aient été mis à jour.

```
---
- name: Rolling update
hosts: webservers
serial: 2
tasks:
- name: latest apache httpd package is installed
  yum:
    name: httpd
    state: latest
  notify: restart apache

handlers:
- name: restart apache
  service:
    name: httpd
    state: restarted
```

Supposons que le groupe `webservers` dans l'exemple précédent contienne cinq serveurs Web situés derrière un équilibrEUR de charge. Avec le paramètre `serial` défini sur **2**, le play ne fonctionnera que sur deux serveurs Web à la fois. La majorité des cinq serveurs Web seront toujours disponibles.

En revanche, en l'absence de mot-clé `serial`, l'exécution du play et donc l'exécution du gestionnaire se produiront simultanément sur les cinq serveurs Web. Cela entraînerait probablement une panne de service, car les services Web seraient redémarrés en même temps sur tous les serveurs Web.

**IMPORTANT**

À certaines fins, chaque lot d'hôtes compte comme s'il s'agissait d'un play complet s'exécutant sur un sous-ensemble d'hôtes. Cela signifie que si un lot entier échoue, le play échoue, ce qui entraîne l'échec de l'exécution du playbook complet à ce stade.

C'est utile. Dans le scénario précédent lorsque **serial: 2** est défini, si un problème survient et que le play échoue pour les deux premiers hôtes traités, le playbook sera annulé et les trois hôtes restants ne seront pas exécutés dans le play.

Le mot clé **serial** peut également être spécifié sous forme de pourcentage. Ce pourcentage est appliqué au nombre total d'hôtes du play pour déterminer la taille du lot de mise à jour progressive. Quel que soit le pourcentage, le nombre d'hôtes par passe sera toujours égal ou supérieur à 1.

**RÉFÉRENCES**

**Taille des lots de mise à jour – délégation, mises à jour progressives et actions locales – Documentation Ansible**

[http://docs.ansible.com/ansible/playbooks\\_delegation.html#rolling-update-batch-size](http://docs.ansible.com/ansible/playbooks_delegation.html#rolling-update-batch-size)

**Optimisation des performances d'Ansible (pour le plaisir et le profit)**

<https://www.ansible.com/blog/ansible-performance-tuning>

## ► EXERCICE GUIDÉ

# CONFIGURATION D'UN PARALLÉLISME

Dans cet exercice, vous allez explorer les effets de différentes directives de séries et de forks sur la façon dont un play est traité par Ansible.

## RÉSULTATS

Vous devrez pouvoir accorder des exécutions en parallèle et en série d'un playbook sur plusieurs hôtes gérés.

Connectez-vous à `workstation` en tant qu'utilisateur `student` avec le mot de passe `student`.

Sur `workstation`, exécutez le script `lab projects-parallelism setup`. Le script de configuration vérifie qu'Ansible est installé sur `workstation` et crée la structure de répertoire et les fichiers associés pour l'environnement de l'atelier.

```
[student@workstation ~]$ lab projects-parallelism setup
```

- 1. Sur `workstation`, en tant qu'utilisateur `student`, accédez au répertoire `~/projects-parallelism`.

```
[student@workstation ~]$ cd ~/projects-parallelism
[student@workstation projects-parallelism]$
```

- 2. Examinez le contenu du répertoire de projet pour vous familiariser avec les fichiers du projet.

- 2.1. Examinez le contenu du fichier `ansible.cfg`. Notez que le fichier d'inventaire est défini sur `inventory`. Notez également que le paramètre `forks` est défini sur **4**.

```
[defaults]
inventory=inventory
remote_user=devops
forks=4
...output omitted...
```

- 2.2. Examinez le contenu du fichier `inventory`. Notez qu'il contient un groupe d'hôtes, `webservers`, qui contient quatre hôtes.

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

```
serverd.lab.example.com
```

- 2.3. Examinez le contenu du fichier **playbook.yml**. Le playbook s'exécute sur le groupe d'hôtes `webservers`. Cela permet de s'assurer que le dernier paquetage `httpd` est installé, et que le service `httpd` est activé et démarré.

```
---
- name: Update web server
  hosts: webservers

  tasks:
    - name: Lastest httpd package installed
      yum:
        name: httpd
        state: latest
      notify:
        - Restart httpd

  handlers:
    - name: Restart httpd
      service:
        name: httpd
        enabled: yes
        state: restarted
```

- 2.4. Pour fin, examinez le contenu du fichier **remove\_apache.yml**. Le playbook s'exécute sur le groupe d'hôtes `webservers`. Cela permet de s'assurer que le service `httpd` est désactivé et arrêté, puis que le paquetage `httpd` n'est pas installé.

```
---
- hosts: webservers
  tasks:
    - service:
        name: httpd
        enabled: no
        state: stopped
    - yum:
        name: httpd
        state: absent
```

- 3. Exécutez le playbook **playbook.yml**, en utilisant la commande `time` pour déterminer le temps qu'il faut au playbook pour s'exécuter. Regardez le playbook pendant son exécution. Notez comment Ansible effectue chaque tâche sur les quatre hôtes en même temps.

```
[student@workstation projects-parallelism]$ time ansible-playbook playbook.yml
PLAY [Update apache] ****
TASK [Gathering Facts] ****
ok: [servera.lab.example.com]
ok: [serverd.lab.example.com]
ok: [serverb.lab.example.com]
ok: [serverc.lab.example.com]
```

```
...output omitted...

real    0m22.701s
user    0m23.275s
sys     0m2.637s
```

- ▶ 4. Exécutez le playbook **remove\_apache.yml** pour arrêter et désactiver le service `httpd` et supprimer le paquetage `httpd`.

```
[student@workstation projects-parallelism]$ ansible-playbook remove_apache.yml
```

- ▶ 5. Changez la valeur du paramètre `forks` par **2** dans **ansible.cfg**.

```
[defaults]
inventory=inventory
remote_user=devops
forks=2
...output omitted...
```

- ▶ 6. Exécutez à nouveau le playbook **playbook.yml**, en utilisant la commande **time** pour déterminer le temps qu'il faut au playbook pour s'exécuter. Regardez le playbook pendant son exécution. Notez que cette fois, Ansible exécute chaque tâche sur deux hôtes seulement, puis sur les deux autres hôtes. Notez également que la diminution du nombre de forks a ralenti l'exécution du playbook.

```
[student@workstation projects-parallelism]$ time ansible-playbook playbook.yml

PLAY [Update apache] ****
TASK [Gathering Facts] ****
ok: [serverb.lab.example.com]
ok: [servera.lab.example.com]
ok: [serverc.lab.example.com]
ok: [serverd.lab.example.com]

...output omitted...

real    0m37.853s
user    0m22.414s
sys     0m4.749s
```

- ▶ 7. Exécutez le playbook **remove\_apache.yml** pour arrêter et désactiver le service `httpd` et supprimer le paquetage `httpd`.

```
[student@workstation projects-parallelism]$ ansible-playbook remove_apache.yml
```

- 8. Ajoutez le paramètre **serial** suivant au play dans le playbook **playbook.yml** afin que le play ne s'exécute que sur deux hôtes à la fois. Le début du playbook doit se présenter comme suit :

```
---  
- name: Update web server  
  hosts: webservers  
  serial: 2
```

- 9. Exécutez à nouveau le playbook **playbook.yml**. Regardez le playbook pendant son exécution. Notez comment Ansible exécute le play entier sur seulement deux hôtes avant ré-exécuter le play sur les deux hôtes restants.

```
[student@workstation projects-parallelism]$ ansible-playbook playbook.yml  
  
PLAY [Update apache] ****  
  
TASK [Gathering Facts] ****  
ok: [servera.lab.example.com]  
ok: [serverb.lab.example.com]  
  
TASK [Latest version of apache installed] ****  
changed: [servera.lab.example.com]  
changed: [serverb.lab.example.com]  
  
...output omitted...  
  
PLAY [Update apache] ****  
  
TASK [Gathering Facts] ****  
ok: [serverc.lab.example.com]  
ok: [serverd.lab.example.com]  
  
TASK [Latest version of apache installed] ****  
changed: [serverd.lab.example.com]  
changed: [serverc.lab.example.com]  
  
...output omitted...
```

- 10. Exécutez le playbook **remove\_apache.yml** pour arrêter et désactiver le service `httpd` et supprimer le paquetage `httpd`.

```
[student@workstation projects-parallelism]$ ansible-playbook remove_apache.yml
```

- 11. Définissez le paramètre **serial** dans le playbook **playbook.yml** sur 3. Le début du playbook doit se présenter comme suit :

```
---  
- name: Update web server  
  hosts: webservers  
  serial: 3
```

- 12. Exécutez à nouveau le playbook **playbook.yml**. Regardez le playbook pendant son exécution. Notez comment Ansible exécute le play entier sur seulement trois hôtes puis ré-exécute le play sur l'hôte restant.

```
[student@workstation projects-parallelism]$ ansible-playbook playbook.yml

PLAY [Update apache] ****

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

TASK [Latest version of apache installed] ****
changed: [servera.lab.example.com]
changed: [serverb.lab.example.com]
changed: [serverc.lab.example.com]

...output omitted...

PLAY [Update apache] ****

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

TASK [Latest version of apache installed] ****
changed: [serverd.lab.example.com]

...output omitted...
```

## Nettoyage

Sur workstation, exédez le script **lab projects-parallelism cleanup** pour effacer l'exercice.

```
[student@workstation ~]$ lab projects-parallelism cleanup
```

L'exercice guidé est maintenant terminé.

# INCLUSION ET IMPORTATION DE FICHIERS

---

Au terme de cette section, les stagiaires doivent pouvoir gérer des playbooks volumineux en important ou en incluant d'autres playbooks ou tâches à partir de fichiers externes, de manière inconditionnelle ou sur la base d'un test conditionnel.

## GESTION DES PLAYBOOKS VOLUMINEUX

Lorsqu'un playbook devient long ou complexe, vous pouvez le diviser en fichiers plus petits pour faciliter sa gestion. Vous pouvez combiner plusieurs playbooks en un playbook principal de manière modulaire ou insérer des listes de tâches d'un fichier dans un play. Cela peut faciliter la réutilisation de plays ou de séquences de tâches dans différents projets.

## INCLUSION ET IMPORTATION DE FICHIERS

Ansible peut utiliser deux opérations pour importer du contenu dans un playbook. Vous pouvez *include* du contenu, ou vous pouvez *importer* du contenu.

Lorsque vous incluez du contenu, c'est une opération *dynamique*. Les processus compatibles incluent du contenu lors de l'exécution du playbook, à mesure que le contenu est atteint.

Lorsque vous importez du contenu, c'est une opération *statique*. Ansible pré-traite le contenu importé lors de l'analyse initiale du playbook, avant le début de l'exécution.

## IMPORTATION DE PLAYBOOKS

La directive `import_playbook` vous permet d'importer des fichiers externes contenant des listes de plays dans un playbook. En d'autres termes, elle vous permet d'avoir un playbook maître qui importe un ou plusieurs playbooks supplémentaires en lui-même.

Le contenu importé étant un playbook complet, la fonction `import_playbook` ne peut être utilisée qu'au plus haut niveau d'un playbook et ne peut pas être utilisée dans un play. Si vous importez plusieurs playbooks, ils seront importés et exécutés dans l'ordre.

Voici un exemple simple de playbook maître qui importe deux playbooks supplémentaires :

```
- name: Prepare the web server
  import_playbook: web.yml

- name: Prepare the database server
  import_playbook: db.yml
```

Vous pouvez également entrelacer des plays dans votre playbook maître avec des playbooks importés.

```
- name: Play 1
  hosts: localhost
  tasks:
    - debug:
        msg: Play 1
```

```
- name: Import Playbook
  import_playbook: play2.yml
```

Dans l'exemple précédent, le **Play 1** s'exécute d'abord, puis les plays importés du playbook **play2.yml**.

## IMPORTATION ET INCLUSION DE TÂCHES

Vous pouvez importer ou inclure une liste de tâches d'un fichier de tâches dans un play. Un fichier de tâches est un fichier contenant une liste de tâches :

```
[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
```

### Importation de fichiers de tâches

Vous pouvez importer de manière statique un fichier de tâches dans un play à l'aide de la fonction **import\_tasks**. Lorsque vous importez un fichier de tâches, les tâches de ce fichier sont directement insérées lors de l'analyse du playbook. L'emplacement de **import\_tasks** dans le playbook détermine l'endroit où les tâches sont insérées et l'ordre dans lequel plusieurs importations sont exécutées.

```
---
- name: Install web server
  hosts: webservers
  tasks:
    - import_tasks: webserver_tasks.yml
```

Lorsque vous importez un fichier de tâches, les tâches de ce fichier sont directement insérées lors de l'analyse du playbook. Parce que **import\_tasks** importe statiquement les tâches lorsque le playbook est analysé, cela a certains effets sur son fonctionnement.

- Lorsque vous utilisez la fonction **import\_tasks**, des instructions conditionnelles telles que **when** définies lors de l'importation sont appliquées à chacune des tâches importées.
- Les boucles ne peuvent pas être utilisées avec **import\_tasks**.
- Si vous utilisez une variable pour spécifier le nom du fichier à importer, vous ne pouvez pas utiliser une variable d'inventaire d'hôte ou de groupe.

### Inclusion de fichiers de tâches

Vous pouvez inclure de manière dynamique un fichier de tâches dans un playbook à l'aide de la fonction **include\_tasks**.

```
---  
- name: Install web server  
  hosts: webservers  
  tasks:  
    - include_tasks: webserver_tasks.yml
```

La fonction `include_tasks` ne traite pas le contenu du playbook tant que le play n'est pas en cours d'exécution et que cette partie du play n'a pas été atteinte. Cela a des conséquences sur son fonctionnement.

- Lorsque vous utilisez la fonction `include_tasks`, des instructions conditionnelles telles que `when` définies sur l'inclusion déterminent si les tâches sont ou non incluses dans le play.
- Si vous exécutez `ansible-playbook --list-tasks` pour lister les tâches du playbook, les tâches dans les fichiers de tâches inclus ne sont pas affichées. Les tâches qui incluent les fichiers de tâches sont affichées. (La fonction `import_tasks`, en comparaison, ne liste pas les tâches qui importent des fichiers de tâches, mais plutôt les tâches individuelles provenant des fichiers de tâches importés.)
- Vous ne pouvez pas utiliser `ansible-playbook --start-at-task` pour démarrer l'exécution du playbook à partir d'une tâche figurant dans un fichier de tâches inclus.
- Vous ne pouvez pas utiliser une instruction `notify` pour déclencher un nom de gestionnaire figurant dans un fichier de tâches inclus. Vous pouvez déclencher un gestionnaire dans le playbook principal contenant un fichier de tâches complet. Dans ce cas, toutes les tâches du fichier inclus seront exécutées.



#### NOTE

Vous trouverez une discussion plus détaillée sur les différences de comportement entre `import_tasks` et `include_tasks` lorsque des conditions sont utilisées sur « Conditions » [[https://docs.ansible.com/ansible/latest/user\\_guide/playbooks\\_conditionals.html#applying-when-to-roles-imports-and-includes](https://docs.ansible.com/ansible/latest/user_guide/playbooks_conditionals.html#applying-when-to-roles-imports-and-includes)] dans le *Guide de l'utilisateur Ansible*.

## Cas d'utilisation pour les fichiers de tâches

Dans les exemples suivants, il peut être utile de gérer des ensembles de tâches en tant que fichiers distincts du playbook :

- Si de nouveaux serveurs imposent une configuration complète, les administrateurs peuvent créer plusieurs ensembles de tâches pour créer des utilisateurs, installer des paquetages, configurer des services et des privilèges, définir les accès à un système de fichiers partagé, renforcer les serveurs, installer les mises à jour de sécurité et un agent de surveillance. Chaque ensemble de tâches peut être géré à l'aide d'un fichier de tâches distinct ayant son propre conteneur.
- Si des serveurs sont gérés collectivement par les développeurs, les administrateurs système et les administrateurs de base de données, alors chaque entité peut écrire son propre fichier de tâches qui peut ensuite être examiné et intégré par le gestionnaire de systèmes.
- Si un serveur nécessite une configuration particulière, celle-ci peut être intégrée sous forme d'ensemble de tâches exécutées sur une condition (à savoir, inclure les tâches uniquement si les critères définis sont remplis).

- Si un groupe de serveurs a besoin d'exécuter une tâche ou un ensemble de tâches particulières, il est possible de les exécuter sur un serveur à la seule condition que le serveur fasse partie d'un groupe d'hôtes particulier.

## Gestion des fichiers de tâches

Vous pouvez créer un répertoire dédié pour les fichiers de tâches, et enregistrer tous les fichiers de tâches dans ce répertoire. Ensuite, votre playbook peut simplement inclure ou importer des fichiers de tâches à partir de ce répertoire. Cela permet de construire un playbook complexe dont la structure et les composants sont faciles à gérer.

## DÉFINITION DE VARIABLES POUR LES PLAYS ET LES TÂCHES EXTERNES

L'intégration de plays ou de tâches à partir de fichiers externes dans les playbooks à l'aide des fonctions d'importation et d'inclusion d'Ansible améliore considérablement la possibilité de réutiliser des tâches et des playbooks dans un environnement Ansible. Pour maximiser les possibilités de réutilisation, ces fichiers de tâche et de play doivent être aussi génériques que possible. Les variables peuvent être utilisées pour paramétriser les éléments de play et de tâche afin d'élargir le champ d'application des tâches et des plays.

Par exemple, le fichier de tâches suivant installe le paquetage requis pour un service Web, puis active et démarre le service nécessaire.

```
---
- name: Install the httpd package
  yum:
    name: httpd
    state: latest
- name: Start the httpd service
  service:
    name: httpd
    enabled: true
    state: started
```

Si vous paramétrez les éléments de paquetage et de service comme indiqué dans l'exemple suivant, alors le fichier de tâche peut également être utilisé pour l'installation et l'administration d'autres logiciels et de leurs services, plutôt que d'être utile pour le service Web uniquement.

```
---
- name: Install the {{ package }} package
  yum:
    name: "{{ package }}"
    state: latest
- name: Start the {{ service }} service
  service:
    name: "{{ service }}"
    enabled: true
    state: started
```

Par la suite, lors de l'intégration du fichier de tâches dans un playbook, vous définissez les variables à utiliser pour l'exécution de la tâche comme suit :

```
...output omitted...
tasks:
- name: Import task file and set variables
  import_tasks: task.yml
vars:
  package: httpd
  service: service
```

Ansible rend les variables transmises disponibles pour les tâches importées à partir du fichier externe.

Vous pouvez également utiliser la même technique pour rendre les fichiers de play plus réutilisables. Lors de l'intégration d'un fichier de play dans un playbook, vous transmettez les variables à utiliser pour l'exécution du play comme suit :

```
...output omitted...
- name: Import play file and set the variable
  import_playbook: play.yml
vars:
  package: mariadb
```



### IMPORTANT

Les versions antérieures d'Ansible utilisaient une fonction `include` pour inclure les playbooks et les fichiers de tâches, en fonction du contexte. Elle est déconseillée pour un certain nombre de raisons.

Avant Ansible 2.0, `include` fonctionnait comme une importation statique. Dans Ansible 2.0, il a été modifié pour fonctionner de manière dynamique, mais cela a parfois entraîné des limitations. Dans Ansible 2.1, il est devenu possible pour `include` d'être dynamique ou statique en fonction des paramètres de la tâche. C'était déroutant et susceptible de générer des erreurs. Il y avait aussi des problèmes pour s'assurer que `include` fonctionnait correctement dans tous les contextes.

À cause de cela, `include` a été remplacé dans Ansible 2.4 par de nouvelles directives telles que `include_tasks`, `import_tasks` et `import_playbook`. Vous trouverez peut-être des exemples de `include` dans des playbooks plus anciens, mais évitez de les utiliser dans les nouveaux.



### RÉFÉRENCES

#### Inclusion et importation – Documentation Ansible

[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks\\_reuse/includes.html](https://docs.ansible.com/ansible/2.7/user_guide/playbooks_reuse/includes.html)

#### Création de playbooks réutilisables – Documentation Ansible

[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks\\_reuse.html](https://docs.ansible.com/ansible/2.7/user_guide/playbooks_reuse.html)

#### Conditions – Documentation Ansible

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

## ► EXERCICE GUIDÉ

# INCLUSION ET IMPORTATION DE FICHIERS

Dans cet exercice, vous allez inclure et importer des playbooks et des tâches dans un playbook Ansible de niveau supérieur.

## RÉSULTATS

Vous devez pouvoir inclure des fichiers de tâches et de playbooks dans des playbooks.

Sur **workstation**, exécutez le script de configuration de l'atelier pour vérifier que l'environnement est prêt pour le début de l'atelier. Le script crée le répertoire de travail, **/home/student/projects-file**, ainsi que les fichiers de projet associés.

```
[student@workstation ~]$ lab projects-file setup
```

- 1. Sur **workstation**, en tant qu'utilisateur **student**, accédez au répertoire **~/projects-file**.

```
[student@workstation ~]$ cd ~/projects-file
[student@workstation projects-file]$
```

- 2. Passez en revue le contenu des trois fichiers dans le sous-répertoire **tasks**.

- 2.1. Parcourez le contenu du fichier **tasks/environment.yml**. Le fichier contient des tâches pour l'installation du paquetage et l'administration du service.

```
[student@workstation projects-file]$ cat tasks/environment.yml
---
- name: Install the {{ package }} package
  yum:
    name: "{{ package }}"
    state: latest
- name: Start the {{ service }} service
  service:
    name: "{{ service }}"
    enabled: true
    state: started
```

- 2.2. Parcourez le contenu du fichier **tasks/firewall.yml**. Le fichier contient les tâches d'installation, d'administration et de configuration du logiciel de pare-feu.

```
[student@workstation projects-file]$ cat tasks/firewall.yml
---
- 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: "{{ rule }}"
    immediate: true
    permanent: true
    state: enabled

```

- 2.3. Parcourez le contenu du fichier **tasks/placeholder.yml**. Le fichier contient une tâche permettant de renseigner un fichier de contenu Web faisant office d'espace réservé.

```
[student@workstation projects-file]$ cat tasks/placeholder.yml
---
- name: Create placeholder file
  copy:
    content: "{{ ansible_facts['fqdn'] }} has been customized using
Ansible. }\n"
    dest: "{{ file }}"

```

- 3. Passez en revue le contenu du fichier **test.yml** dans le sous-répertoire **plays**. Le fichier contient un play qui teste les connexions à un service Web.

```

---
- name: Test web service
  hosts: localhost
  become: no
  tasks:
    - name: connect to internet web server
      uri:
        url: "{{ url }}"
        status_code: 200

```

- 4. Créez un playbook nommé **playbook.yml**. Définissez le premier play avec le nom **Configurer le serveur Web**. Le play doit être exécuté sur les hôtes gérés de `servera.lab.example.com` définis dans le fichier **inventory**. Le début du fichier doit ressembler à ceci :

```

---
- name: Configure web server
  hosts: servera.lab.example.com

```

- 5. Dans le playbook **playbook.yml**, définissez la section des tâches avec trois ensembles de tâches. Importez le premier jeu de tâches à partir du fichier de tâches **tasks/environment.yml** et définissez les variables nécessaires pour installer le paquetage **httpd**, et pour activer et démarrer le service **ht tpd**. Importez le deuxième jeu de tâches à partir du fichier de tâches **tasks/firewall.yml** et définissez les variables nécessaires pour installer le paquetage **firewalld** pour activer et démarrer le service **firewalld**, et pour autoriser les connexions **http**. Importez le troisième ensemble de tâches à partir du fichier de tâches **tasks/placeholder.yml**.

- 5.1. Créez la section des tâches dans le premier play en ajoutant l'entrée suivante au playbook **playbook.yml**.

```
tasks:
```

- 5.2. Importez le premier ensemble de tâches depuis **tasks/environment.yml** en utilisant la fonction **import\_tasks**. Définissez **httpd** comme valeur pour les variables **package** et **service**. Définissez **started** comme valeur pour la variable **svc\_state**.

```
- name: Import the environment task file and set the variables
  import_tasks: tasks/environment.yml
  vars:
    package: httpd
    service: httpd
```

- 5.3. Importez le deuxième ensemble de tâches depuis **tasks/firewall.yml** en utilisant la fonction **import\_tasks**. Définissez **firewalld** comme valeur pour les variables **firewall\_pkg** et **firewall\_svc**. Définissez **http** comme valeur pour la variable **rule**.

```
- name: Import the firewall task file and set the variables
  import_tasks: tasks/firewall.yml
  vars:
    firewall_pkg: firewalld
    firewall_svc: firewalld
    rule: http
```

- 5.4. Importez le dernier ensemble de tâches depuis **tasks/placeholder.yml** en utilisant la fonction **import\_tasks**. Définissez **/var/www/html/index.html** comme valeur pour la variable **file**.

```
- name: Import the placeholder task file and set the variable
  import_tasks: tasks/placeholder.yml
  vars:
    file: /var/www/html/index.html
```

- 6. Ajoutez un deuxième et dernier play au playbook **playbook.yml** en utilisant le contenu du playbook **plays/test.yml**.

- 6.1. Ajoutez un deuxième play au playbook **playbook.yml** pour valider l'installation du serveur Web. Importez le play à partir de **plays/test.yml**. Définissez **http://servera.lab.example.com** comme valeur pour la variable **url**.

```
- name: Import test play file and set the variable
  import_playbook: plays/test.yml
  vars:
    url: 'http://servera.lab.example.com'
```

- 6.2. Votre playbook ressemble à ce qui suit une fois les modifications terminées :

```
---
- name: Configure web server
  hosts: servera.lab.example.com

  tasks:
    - name: Import the environment task file and set the variables
      import_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
    - 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. Enregistrez les modifications dans le playbook **playbook.yml**.

- 7. Avant d'exécuter le playbook, vérifiez que sa syntaxe est correcte à l'aide de la commande **ansible-playbook --syntax-check**. En cas d'erreurs, corrigez-les avant de passer à l'étape suivante.

```
[student@workstation projects-file]$ ansible-playbook playbook.yml --syntax-check
playbook: playbook.yml
```

- 8. Exécutez le playbook **playbook.yml**. La sortie du playbook montre l'importation de la tâche et des fichiers de play.

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

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=7    changed=1    unreachable=0   failed=0
```

## Nettoyage

Sur workstation, exéutez le script **lab projects-file cleanup** pour effacer l'exercice.

```
[student@workstation ~]$ lab projects-file cleanup
```

L'exercice guidé est maintenant terminé.

## ► OPEN LAB

# GESTION DE PROJETS VOLUMINEUX

## LISTE DE CONTRÔLE DES PERFORMANCES

Au cours de cet atelier, vous allez modifier un playbook volumineux pour qu'il soit plus facile à gérer en utilisant des modèles d'hôtes, des inclusions, des importations et un inventaire dynamique, et vous ajusterez le mode de traitement du playbook par Ansible.

## RÉSULTATS

Vous devez pouvoir simplifier les références d'hôte gérées dans un playbook en spécifiant des modèles d'hôte par rapport à un inventaire dynamique. Vous devez également pouvoir restructurer un playbook pour que les tâches soient importées à partir de fichiers de tâches externes et le configurer pour des mises à jour progressives.

Connectez-vous en tant qu'utilisateur `student` sur `workstation`, puis exécutez `lab projects-review setup`. Ce script de configuration permet de s'assurer que les hôtes gérés sont accessibles sur le réseau. Il garantit également que le fichier de configuration Ansible, le fichier d'inventaire et le playbook appropriés sont installés sur le nœud de contrôle.

```
[student@workstation ~]$ lab projects-review setup
```

Vous avez hérité d'un playbook de l'administrateur précédent. Le playbook est utilisé pour configurer le service Web sur `servera.lab.example.com`, `serverb.lab.example.com`, `serverc.lab.example.com` et `serverd.lab.example.com`. Le playbook configure également le pare-feu sur les quatre hôtes gérés afin que le trafic Web soit autorisé.

Apportez les modifications suivantes au fichier de playbook `playbook.yml` de manière à ce qu'il soit plus facile à gérer et à définir, de sorte que les exécutions futures utilisent des mises à jour progressives pour empêcher les quatre serveurs Web d'être indisponibles en même temps.

1. Simplifiez la liste des hôtes gérés dans le playbook en utilisant un modèle d'hôte générique.
2. Restructurez le playbook de manière à ce que les deux premières tâches du playbook soient conservées dans un fichier de tâches externe situé dans `tasks/web_tasks.yml`. Utilisez la fonction `import_tasks` pour incorporer ce fichier de tâches dans le playbook.
3. Restructurez le playbook de manière à ce que la troisième, la quatrième et la cinquième tâches du playbook soient conservées dans un fichier de tâches externe situé dans `tasks/firewall_tasks.yml`. Utilisez la fonction `import_tasks` pour incorporer ce fichier de tâches dans le playbook.
4. Le gestionnaire de redémarrage du service `httpd` pouvant être déclenché si de futures modifications sont apportées au fichier `files/tune.conf`, implémentez la fonctionnalité de mise à jour progressive dans le playbook `playbook.yml` et définissez la taille du lot de mise à jour progressive sur deux hôtes.
5. Vérifiez que les modifications apportées au playbook `playbook.yml` ont été correctement effectuées, puis exécutez le playbook.

## Évaluation

À partir de `workstation`, exécutez la commande **lab projects-review grade** pour confirmer que l'exercice est réussi. Corrigez toute erreur signalée et répétez le script tant que des erreurs persistent.

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

## Nettoyage

Sur `workstation`, exécutez le script **lab projects-review cleanup** pour effacer les ressources créées au cours de cet atelier.

```
[student@workstation ~]$ lab projects-review cleanup
```

L'atelier est maintenant terminé.

## ► SOLUTION

# GESTION DE PROJETS VOLUMINEUX

### LISTE DE CONTRÔLE DES PERFORMANCES

Au cours de cet atelier, vous allez modifier un playbook volumineux pour qu'il soit plus facile à gérer en utilisant des modèles d'hôtes, des inclusions, des importations et un inventaire dynamique, et vous ajusterez le mode de traitement du playbook par Ansible.

### RÉSULTATS

Vous devez pouvoir simplifier les références d'hôte gérées dans un playbook en spécifiant des modèles d'hôte par rapport à un inventaire dynamique. Vous devez également pouvoir restructurer un playbook pour que les tâches soient importées à partir de fichiers de tâches externes et le configurer pour des mises à jour progressives.

Connectez-vous en tant qu'utilisateur **student** sur **workstation**, puis exécutez **lab projects-review setup**. Ce script de configuration permet de s'assurer que les hôtes gérés sont accessibles sur le réseau. Il garantit également que le fichier de configuration Ansible, le fichier d'inventaire et le playbook appropriés sont installés sur le nœud de contrôle.

```
[student@workstation ~]$ lab projects-review setup
```

Vous avez hérité d'un playbook de l'administrateur précédent. Le playbook est utilisé pour configurer le service Web sur **servera.lab.example.com**, **serverb.lab.example.com**, **serverc.lab.example.com** et **serverd.lab.example.com**. Le playbook configure également le pare-feu sur les quatre hôtes gérés afin que le trafic Web soit autorisé.

Apportez les modifications suivantes au fichier de playbook **playbook.yml** de manière à ce qu'il soit plus facile à gérer et à définir, de sorte que les exécutions futures utilisent des mises à jour progressives pour empêcher les quatre serveurs Web d'être indisponibles en même temps.

1. Simplifiez la liste des hôtes gérés dans le playbook en utilisant un modèle d'hôte générique.
  - 1.1. Examinez le fichier de configuration **ansible.cfg** pour déterminer l'emplacement du fichier d'inventaire. Vous devriez voir que l'inventaire est défini en tant que sous-répertoire **inventory** et que ce sous-répertoire contient un script d'inventaire dynamique **inventory.py**.

```
[student@workstation ~]$ cd ~/projects-review
[student@workstation projects-review]$ cat ansible.cfg
[defaults]
inventory = inventory
...output omitted...
[student@workstation projects-review]$ ll
total 16
-rw-rw-r--. 1 student student 33 Dec 19 00:48 ansible.cfg
drwxrwxr-x. 2 student student 4096 Dec 18 22:35 files
drwxrwxr-x. 2 student student 4096 Dec 19 01:18 inventory
-rw-rw-r--. 1 student student 959 Dec 18 23:48 playbook.yml
```

**CHAPITRE 7** | Gestion de projets volumineux

```
[student@workstation projects-review]$ ll inventory
total 4
-rwxrwxr-x. 1 student student 612 Dec 19 01:18 inventory.py
```

- 1.2. Rendez le script d'inventaire dynamique **inventory/inventory.py** exécutable, puis exécutez le script d'inventaire dynamique avec l'option **--list** pour afficher la liste complète des hôtes dans l'inventaire.

```
[student@workstation projects-review]$ chmod 755 inventory/inventory.py
[student@workstation projects-review]$ inventory/inventory.py --list
{"all": {"hosts": ["servera.lab.example.com", "serverb.lab.example.com",
"serverc.lab.example.com", "serverd.lab.example.com",
"workstation.lab.example.com"], "vars": {}}}
```

- 1.3. Vérifiez que le modèle d'hôte **server\*.lab.example.com** identifie correctement les quatre hôtes gérés qui sont ciblés par le playbook **playbook.yml**.

```
[student@workstation projects-review]$ ansible server*.lab.example.com --list-
hosts
hosts (4):
  serverb.lab.example.com
  serverd.lab.example.com
  servera.lab.example.com
  serverc.lab.example.com
```

- 1.4. Remplacez la liste des hôtes dans le playbook **playbook.yml** par le modèle d'hôte **server\*.lab.example.com**.

```
[student@workstation projects-review]$ cat playbook.yml
---
- name: Install and configure web service
  hosts: server*.lab.example.com
  ...output omitted...
```

2. Restructurez le playbook de manière à ce que les deux premières tâches du playbook soient conservées dans un fichier de tâches externe situé dans **tasks/web\_tasks.yml**. Utilisez la fonction **import\_tasks** pour incorporer ce fichier de tâches dans le playbook.

- 2.1. Créez le sous-répertoire **tasks**.

```
[student@workstation projects-review]$ mkdir tasks
```

- 2.2. Placez le contenu des deux premières tâches du playbook **playbook.yml** dans le fichier **tasks/web\_tasks.yml**. Le fichier de tâche doit contenir les éléments suivants :

```
---
- name: Install httpd
  yum:
    name: httpd
    state: latest
```

```
- 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. Supprimez les deux premières tâches du playbook **playbook.yml** et insérez les lignes suivantes à leur place pour importer le fichier de tâches **tasks/web\_tasks.yml**.

```
- name: Import the web_tasks.yml task file
import_tasks: tasks/web_tasks.yml
```

3. Restructurez le playbook de manière à ce que la troisième, la quatrième et la cinquième tâches du playbook soient conservées dans un fichier de tâches externe situé dans **tasks/firewall\_tasks.yml**. Utilisez la fonction **import\_tasks** pour incorporer ce fichier de tâches dans le playbook.

- 3.1. Placez le contenu des trois tâches restantes du playbook **playbook.yml** dans le fichier **tasks/firewall\_tasks.yml**. Le fichier de tâche doit contenir les éléments suivants.

```
---
- name: Install firewalld
yum:
  name: firewalld
  state: latest

- name: 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. Supprimez les trois tâches restantes du playbook **playbook.yml** et insérez les lignes suivantes à leur place pour importer le fichier de tâches **tasks/firewall\_tasks.yml**.

```
- name: Import the firewall_tasks.yml task file
import_tasks: tasks/firewall_tasks.yml
```

4. Le gestionnaire de redémarrage du service httpd pouvant être déclenché si de futures modifications sont apportées au fichier **files/tune.conf**, implémentez la fonctionnalité de mise à jour progressive dans le playbook **playbook.yml** et définissez la taille du lot de mise à jour progressive sur deux hôtes.

4.1. Ajoutez le paramètre **serial** sur le playbook **playbook.yml**.

```
[student@workstation projects-review]$ cat playbook.yml
---
- name: Install and configure web service
  hosts: server*.lab.example.com
  serial: 2
...output omitted...
```

5. Vérifiez que les modifications apportées au playbook **playbook.yml** ont été correctement effectuées, puis exécutez le playbook.

5.1. Vérifiez que le playbook **playbook.yml** présente le contenu suivant.

```
---
- name: Install and configure web service
  hosts: server*.lab.example.com
  serial: 2

  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. Exécutez le playbook avec **ansible-playbook --syntax-check** pour vérifier qu'il ne contient aucune erreur de syntaxe. Si des erreurs sont présentes, corrigez-les avant.

```
[student@workstation projects-review]$ ansible-playbook playbook.yml --syntax-
check
playbook: playbook.yml
```

- 5.3. Exécutez le playbook. Le playbook doit être exécuté sur l'hôte en tant que mise à jour progressive avec une taille de lot de deux hôtes gérés.

```
[student@workstation projects-review]$ ansible-playbook playbook.yml
PLAY [Install and configure web service] ****
TASK [Gathering Facts] ****
ok: [serverd.lab.example.com]
```

```
ok: [serverb.lab.example.com]

TASK [Install httpd] *****
changed: [serverd.lab.example.com]
changed: [serverb.lab.example.com]

TASK [Tuning configuration installed] *****
changed: [serverb.lab.example.com]
changed: [serverd.lab.example.com]

TASK [Install firewalld] *****
ok: [serverb.lab.example.com]
ok: [serverd.lab.example.com]

TASK [Start the firewall] *****
ok: [serverd.lab.example.com]
ok: [serverb.lab.example.com]

TASK [Open the port for http] *****
changed: [serverd.lab.example.com]
changed: [serverb.lab.example.com]

RUNNING HANDLER [restart httpd] *****
changed: [serverb.lab.example.com]
changed: [serverd.lab.example.com]

PLAY [Install and configure web service] *****

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

TASK [Install httpd] *****
changed: [serverc.lab.example.com]
changed: [servera.lab.example.com]

TASK [Tuning configuration installed] *****
changed: [servera.lab.example.com]
changed: [serverc.lab.example.com]

TASK [Install firewalld] *****
ok: [servera.lab.example.com]
ok: [serverc.lab.example.com]

TASK [Start the firewall] *****
ok: [servera.lab.example.com]
ok: [serverc.lab.example.com]

TASK [Open the port for http] *****
changed: [servera.lab.example.com]
changed: [serverc.lab.example.com]

RUNNING HANDLER [restart httpd] *****
changed: [serverc.lab.example.com]
changed: [servera.lab.example.com]
```

```
PLAY RECAP ****
servera.lab.example.com    : ok=7      changed=2      unreachable=0      failed=0
serverb.lab.example.com    : ok=7      changed=3      unreachable=0      failed=0
serverc.lab.example.com    : ok=7      changed=4      unreachable=0      failed=0
serverd.lab.example.com    : ok=7      changed=4      unreachable=0      failed=0
```

## Évaluation

À partir de `workstation`, exécutez la commande **lab projects-review grade** pour confirmer que l'exercice est réussi. Corrigez toute erreur signalée et répétez le script tant que des erreurs persistent.

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

## Nettoyage

Sur `workstation`, exécutez le script **lab projects-review cleanup** pour effacer les ressources créées au cours de cet atelier.

```
[student@workstation ~]$ lab projects-review cleanup
```

L'atelier est maintenant terminé.

# RÉSUMÉ

---

Dans ce chapitre, vous avez appris les principes suivants :

- Les modèles d'hôte servent à spécifier les hôtes gérés à cibler à l'aide de plays ou de commandes ad hoc.
- Les scripts d'inventaire dynamiques peuvent être utilisés pour générer des listes dynamiques d'hôtes gérés à partir de services d'annuaire ou d'autres sources externes à Ansible.
- Les scripts d'inventaire dynamiques doivent être exécutables et doivent renvoyer les informations d'inventaire au format JSON lorsque l'`--list` option leur est transmise.
- Le paramètre `forks` dans le fichier de configuration Ansible définit le nombre maximal de connexions parallèles aux hôtes gérés.
- Le paramètre `serial` peut être utilisé pour implémenter des mises à jour progressives sur des hôtes gérés en définissant le nombre d'hôtes gérés dans chaque lot de mises à jour progressives.
- Vous pouvez utiliser la fonction `import_playbook` pour incorporer des fichiers play externes dans des playbooks.
- Vous pouvez utiliser les fonctions `include_tasks` et `import_tasks` pour incorporer des fichiers de tâche externes dans des playbooks.
- Les fonctionnalités d'importation sont statiques dans leur fonctionnement et prennent effet lorsque le playbook est analysé. Les fonctionnalités d'inclusion sont dynamiques et prennent effet lorsque cette partie du playbook est exécutée.

## CHAPITRE 8

# SIMPLIFICATION DES PLAYBOOKS AVEC DES RÔLES

### PROJET

Utiliser les rôles Ansible pour développer plus rapidement des playbooks et réutiliser le code Ansible.

### OBJECTIFS

- Décrire ce qu'est un rôle, comment il est structuré et comment vous pouvez l'utiliser dans un playbook.
- Créer un rôle dans le répertoire de projet d'un playbook et l'exécuter dans le cadre de l'un des plays du playbook.
- Sélectionner et récupérer des rôles à partir d'Ansible Galaxy ou d'autres sources, telles qu'un référentiel Git, et les utiliser dans vos playbooks.
- Écrire des playbooks qui tirent parti des rôles système Red Hat Enterprise Linux pour effectuer des opérations standard.

### SECTIONS

- Description de la structure des rôles (avec quiz)
- Crédit de rôles (et exercice guidé)
- Déploiement de rôles avec Ansible Galaxy (et exercice guidé)
- Réutilisation de contenu avec des rôles système (et exercice guidé)

### ATELIER

- Simplification des playbooks avec des rôles

# DESCRIPTION DE LA STRUCTURE D'UN RÔLE

---

## OBJECTIFS

Au terme de cette section, les stagiaires doivent pouvoir décrire ce qu'est un rôle, comment il est structuré et comment vous pouvez l'utiliser dans un playbook.

## DÉFINITION D'UNE STRUCTURE POUR LES PLAYBOOKS ANSIBLE À L'AIDE DE RÔLES

Au fur et à mesure que vous développerez plus de playbooks, vous découvrirez probablement que vous avez de nombreuses possibilités de réutiliser le code des playbooks que vous avez déjà écrits. Peut-être qu'un play pour configurer une base de données MySQL pour une application pourrait être réaffecté, avec des noms d'hôte, des mots de passe et des utilisateurs différents, pour configurer une base de données MySQL pour une autre application.

Mais dans le monde réel, ce play peut être long et complexe, avec de nombreux fichiers inclus ou importés, et avec des tâches et des gestionnaires permettant de gérer diverses situations. Copier tout ce code dans un autre playbook peut être une tâche non triviale.

Les *rôles* Ansible vous permettent de réutiliser plus facilement le code Ansible de manière générique. Vous pouvez conditionner, dans une structure de répertoire normalisée, tous les tâches, variables, fichiers, modèles et autres ressources nécessaires pour déployer l'infrastructure ou des applications. Vous pouvez copier ce rôle d'un projet à l'autre simplement en copiant le répertoire de ce rôle. Vous pouvez ensuite simplement appeler ce rôle depuis un play pour l'exécuter.

Un rôle bien écrit vous permettra de transmettre des variables au rôle à partir du playbook qui ajustent son comportement, en définissant tous noms d'hôte, adresses IP, noms d'utilisateur, secrets spécifiques au site ou autres détails spécifiques au site local dont vous avez besoin. Par exemple, un rôle permettant de déployer un serveur de base de données peut avoir été écrit pour prendre en charge des variables définissant le nom d'hôte, l'utilisateur et le mot de passe de l'administrateur de base de données, ainsi que d'autres paramètres nécessitant une personnalisation pour votre installation. L'auteur du rôle peut également s'assurer que des valeurs par défaut raisonnables sont définies pour ces variables si vous choisissez de ne pas les définir dans le play.

Les rôles Ansible présentent les avantages suivants :

- Les rôles permettent de grouper du contenu et facilitent ainsi le partage de code
- Les rôles peuvent être écrits de sorte à définir les éléments essentiels d'un type de système : serveur Web, serveur de base de données, référentiel Git ou autre
- Les rôles facilitent la gestion des projets volumineux
- Les rôles peuvent être développés en parallèle par plusieurs administrateurs

En plus d'écrire, d'utiliser, de réutiliser et de partager vos propres rôles, vous pouvez obtenir des rôles d'autres sources. Certains rôles sont inclus dans Red Hat Enterprise Linux, dans le paquetage *rhel-system-roles*. Vous pouvez également obtenir un grand nombre de rôles pris en charge par la communauté à partir du site Web Ansible Galaxy. Plus tard dans ce chapitre, vous en apprendrez plus sur ces rôles.

## ANALYSE DE LA STRUCTURE DES RÔLES ANSIBLE

Un rôle Ansible est défini par une structure normalisée de sous-répertoires et de fichiers. Le répertoire de niveau supérieur définit le nom du rôle lui-même. Les fichiers sont organisés en sous-répertoires nommés en fonction de l'objectif de chaque fichier dans le rôle, tels que **tasks** et **handlers**. Les sous-répertoires **files** et **templates** contiennent des fichiers auxquels les tâches d'autres fichiers YAML font référence.

La commande **tree** suivante affiche la structure de répertoire du rôle **user.example**.

```
[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
```

### Sous-répertoires des rôles Ansible

| SOUS-RÉPERTOIRE  | FONCTION                                                                                                                                                                                                                                                                     |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>defaults</b>  | Le fichier <b>main.yml</b> de ce répertoire contient les valeurs par défaut des variables de rôle. Celles-ci peuvent être remplacées lorsque le rôle est utilisé. Ces variables ont une faible priorité et sont destinées à être modifiées et personnalisées dans les plays. |
| <b>files</b>     | Le répertoire contient des fichiers statiques auxquels les tâches du rôle font référence.                                                                                                                                                                                    |
| <b>handlers</b>  | Le fichier <b>main.yml</b> de ce répertoire contient les définitions du gestionnaire de rôle.                                                                                                                                                                                |
| <b>meta</b>      | Le fichier <b>main.yml</b> de ce répertoire contient des informations sur le rôle, parmi lesquelles l'auteur, la licence, les plateformes et les dépendances en option du rôle.                                                                                              |
| <b>tasks</b>     | Le fichier <b>main.yml</b> de ce répertoire contient les définitions des tâches du rôle.                                                                                                                                                                                     |
| <b>templates</b> | Le répertoire contient des modèles Jinja2 auxquels les tâches du rôle font référence.                                                                                                                                                                                        |

| SOUS-RÉPERTOIRE | FONCTION                                                                                                                                                                                                                                                                                       |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>tests</b>    | Ce répertoire peut contenir un inventaire et un playbook <b>test.yml</b> qui peuvent être utilisés pour tester le rôle.                                                                                                                                                                        |
| <b>vars</b>     | Le fichier <b>main.yml</b> de ce répertoire définit les valeurs des variables du rôle. Ces variables sont souvent utilisées à des fins internes au sein du rôle. Ces variables ont une priorité élevée et ne sont pas destinées à être modifiées lorsqu'elles sont utilisées dans un playbook. |

Tous les rôles n'auront pas tous ces répertoires.

## DÉFINITION DES VARIABLES ET DES VALEURS PAR DÉFAUT

Les *variables de rôle* sont définies au moyen d'un fichier **vars/main.yml** contenant des paires clé: valeur dans la hiérarchie du répertoire de rôle. Comme pour toutes autres variables, des références à ces variables sont contenues dans le fichier YAML du rôle : `{{ VAR_NAME }}`. Ces variables ont une priorité élevée ; il est impossible de les remplacer par des variables d'inventaire. L'intention de ces variables est qu'elles soient utilisées par le fonctionnement interne du rôle.

Les *variables par défaut* autorisent la définition de valeurs par défaut pour les variables pouvant être utilisées dans un play pour configurer le rôle ou personnaliser son comportement. Elles sont définies au moyen d'un fichier **defaults/main.yml** contenant les paires clé: valeur dans la hiérarchie du répertoire de rôle. De toutes les variables disponibles, les variables par défaut ont la priorité la plus faible. Elles peuvent être facilement remplacées par d'autres variables, y compris des variables d'inventaire. Ces variables sont destinées à fournir à la personne qui écrit un play utilisant ce rôle un moyen de personnaliser ou de contrôler exactement ce qu'il va faire. Elles peuvent être utilisées pour fournir des informations sur le rôle nécessaires pour configurer ou déployer quelque chose correctement.

Définissez une variable spécifique dans **vars/main.yml** ou **defaults/main.yml**, mais pas simultanément. Utilisez des variables par défaut lorsque les valeurs correspondantes doivent être remplacées.



### IMPORTANT

Les rôles ne devraient pas contenir de données spécifiques aux sites. Ils ne doivent absolument pas contenir de secrets tels que des mots de passe ou des clés privées.

En effet, les rôles sont supposés être génériques, réutilisables et pouvoir être partagés librement. Les détails spécifiques aux sites ne doivent pas être codés en dur.

Les secrets doivent être fournis au rôle par d'autres moyens. C'est l'une des raisons pour lesquelles vous souhaiterez peut-être définir des variables de rôle en appelant un rôle. Les variables de rôle définies dans le play peuvent fournir le secret, ou pointer vers un fichier chiffré par Ansible Vault contenant le secret.

## UTILISATION DE RÔLES ANSIBLE DANS UN PLAYBOOK

L'utilisation de rôles dans un playbook est simple. L'exemple suivant indique une façon d'appeler des rôles Ansible.

```
---
- hosts: remote.example.com
  roles:
    - role1
    - role2
```

Pour chaque rôle spécifié, les tâches du rôle, ses gestionnaires, ses variables et ses dépendances seront importées dans le playbook, dans cet ordre. Toute tâche `copy`, `script`, `template` ou `include_tasks/import_tasks` du rôle peut faire référence aux fichiers, modèles ou fichiers tâches pertinents dans le rôle sans nom de chemin relatif ou absolu. Ansible les cherche, respectivement, dans les sous-répertoires `files`, `templates` ou `tasks` du rôle.

Quand vous utilisez une section `roles` pour importer des rôles dans un play, les rôles sont exécutés en premier, avant toutes les tâches que vous définissez pour ce play.

L'exemple suivant définit les valeurs de deux variables de rôle de `role2`. `role1` est utilisé de la même manière que dans l'exemple précédent. Toutes les variables `defaults` et `vars` sont remplacées lorsque `role2` est utilisé.

```
---
- hosts: remote.example.com
  roles:
    - role: role1
    - role: role2
      var1: val1
      var2: val2
```

Une autre syntaxe YAML équivalente que vous pourriez voir dans ce cas est la suivante :

```
---
- hosts: remote.example.com
  roles:
    - role: role1
    - { role: role2, var1: val1, var2: val2 }
```

Il y a des situations où cette syntaxe peut être plus difficile à lire, même si elle est plus compacte.



### IMPORTANT

Les variables de rôle définies en ligne (les paramètres de rôle), comme dans les exemples précédents, présentent une priorité très élevée. Elles remplaceront la plupart des autres variables.

Veillez à ne pas réutiliser les noms des variables de rôle que vous définissez en ligne ailleurs dans votre play, car les valeurs des variables de rôle remplaceront les variables d'inventaire et toute variable `vars` de play.

## CONTRÔLE DE L'ORDRE D'EXÉCUTION

Pour chaque play dans un playbook, les tâches s'exécutent dans l'ordre indiqué dans la liste des tâches. Une fois toutes les tâches exécutées, tous les gestionnaires notifiés sont exécutés.

Lorsqu'un rôle est ajouté à un play, les tâches de rôle sont ajoutées au début de la liste des tâches. Si un second rôle est inclus dans un play, sa liste de tâches est ajoutée après le premier rôle.

Les gestionnaires de rôles sont ajoutés aux plays de la même manière que les tâches de rôles. Chaque play définit une liste de gestionnaires. Les gestionnaires de rôles sont d'abord ajoutés à la liste des gestionnaires, suivis de tous les gestionnaires définis dans la section **handlers** du play.

Dans certains scénarios, il peut être nécessaire d'exécuter certaines tâches play avant les rôles. Pour prendre en charge de tels scénarios, les plays peuvent être configurés avec une section **pre\_tasks**. Toute tâche listée dans cette section s'exécute avant que des rôles ne soient exécutés. Si l'une de ces tâches notifie un gestionnaire, ces tâches de gestionnaire s'exécutent avant les rôles ou les tâches normales.

Les plays prennent également en charge un mot-clé **post\_tasks**. Ces tâches s'exécutent une fois que les tâches normales du play et tous les gestionnaires qu'ils ont notifiés sont exécutés.

Le play suivant montre un exemple avec les éléments **pre\_tasks**, **roles**, **tasks**, **post\_tasks** et **handlers**. Il est inhabituel qu'un play contienne toutes ces sections.

```
- 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
```

Dans l'exemple ci-dessus, une tâche `debug` est exécutée dans chaque section pour notifier le gestionnaire **my handler**. La tâche **my handler** est exécutée trois fois :

- après l'exécution de toutes les tâches **pre\_tasks** ;
- après l'exécution de toutes les tâches de rôle et des tâches de la section **tasks** ;
- après l'exécution de toutes les tâches **post\_tasks**.

Des rôles peuvent être ajoutés au play en utilisant une tâche ordinaire, pas seulement en les incluant dans la section **role** d'un play. Utilisez le module `include_role` pour inclure dynamiquement un rôle et utilisez le module `import_role` pour importer statiquement un rôle.

Le playbook suivant montre comment un rôle peut être inclus à l'aide d'une tâche avec le module `include_role`.

```
- 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
```



### NOTE

Le module `include_role` a été ajouté dans Ansible 2.3, et le module `import_role` a été ajouté dans Ansible 2.4.



### RÉFÉRENCES

#### Rôles – Documentation Ansible

[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks\\_reuse\\_roles.html](https://docs.ansible.com/ansible/2.7/user_guide/playbooks_reuse_roles.html)

## ► QUIZ

# DESCRIPTION DE LA STRUCTURE D'UN RÔLE

Répondez aux questions suivantes en sélectionnant une réponse :

► 1. Parmi les propositions suivantes, laquelle décrit le mieux les rôles ?

- a. Des paramètres de configuration qui permettent à des utilisateurs spécifiques d'exécuter des playbooks Ansible.
- b. Des playbooks pour un datacenter.
- c. Un ensemble de fichiers de tâches YAML et d'éléments connexes organisés en structure afin de faciliter leur partage, leur mobilité et leur réutilisation.

► 2. Parmi les éléments suivants, lesquels peuvent être spécifiés dans les rôles ?

- a. Les gestionnaires
- b. Les tâches
- c. Les modèles
- d. Les variables
- e. Toutes les réponses ci-dessus

► 3. Quel fichier déclare des dépendances de rôle ?

- a. Le playbook Ansible qui utilise le rôle.
- b. Le fichier **meta/main.yml** contenu dans la hiérarchie du rôle.
- c. Le fichier **meta/main.yml** contenu dans le répertoire de projet.
- d. Il est impossible de définir les dépendances d'un rôle dans Ansible.

► 4. Quel fichier d'une structure de répertoire de rôle doit contenir les valeurs initiales des variables pouvant être utilisées comme paramètres du rôle ?

- a. **defaults/main.yml**
- b. **meta/main.yml**
- c. **vars/main.yml**
- d. Le fichier d'inventaire d'hôtes.

## ► SOLUTION

# DESCRIPTION DE LA STRUCTURE D'UN RÔLE

Répondez aux questions suivantes en sélectionnant une réponse :

► 1. Parmi les propositions suivantes, laquelle décrit le mieux les rôles ?

- a. Des paramètres de configuration qui permettent à des utilisateurs spécifiques d'exécuter des playbooks Ansible.
- b. Des playbooks pour un datacenter.
- c. Un ensemble de fichiers de tâches YAML et d'éléments connexes organisés en structure afin de faciliter leur partage, leur mobilité et leur réutilisation.

► 2. Parmi les éléments suivants, lesquels peuvent être spécifiés dans les rôles ?

- a. Les gestionnaires
- b. Les tâches
- c. Les modèles
- d. Les variables
- e. Toutes les réponses ci-dessus

► 3. Quel fichier déclare des dépendances de rôle ?

- a. Le playbook Ansible qui utilise le rôle.
- b. Le fichier `meta/main.yml` contenu dans la hiérarchie du rôle.
- c. Le fichier `meta/main.yml` contenu dans le répertoire de projet.
- d. Il est impossible de définir les dépendances d'un rôle dans Ansible.

► 4. Quel fichier d'une structure de répertoire de rôle doit contenir les valeurs initiales des variables pouvant être utilisées comme paramètres du rôle ?

- a. `defaults/main.yml`
- b. `meta/main.yml`
- c. `vars/main.yml`
- d. Le fichier d'inventaire d'hôtes.

# CRÉATION DE RÔLES

---

## OBJECTIFS

Au terme de cette section, les stagiaires doivent pouvoir créer un rôle dans le répertoire de projet d'un playbook et l'exécuter dans le cadre de l'un des plays du playbook.

## PROCESSUS DE CRÉATION DE RÔLE

Aucun outil de développement spécial n'est requis pour créer des rôles dans Ansible. Créer et utiliser un rôle est une procédure en trois étapes :

1. Créer la structure du répertoire de rôles.
2. Définir le contenu du rôle.
3. Utiliser le rôle dans un playbook.

**Figure 8.0: Crédit de rôle**

## CRÉATION DE LA STRUCTURE DU RÉPERTOIRE DE RÔLES

Par défaut, Ansible recherche les rôles dans un sous-répertoire appelé **roles** dans le répertoire contenant votre playbook Ansible. Cela vous permet de stocker des rôles avec le playbook et d'autres fichiers de prise en charge.

Si Ansible ne peut pas trouver le rôle à cet endroit, il examine les répertoires spécifiés par le paramètre de configuration Ansible **roles\_path**, dans l'ordre. Cette variable contient une liste de répertoires à rechercher, séparés par le signe deux-points. La valeur par défaut de cette variable est la suivante :

```
~/ansible/roles:/usr/share/ansible/roles:/etc/ansible/roles
```

Cela vous permet d'installer sur votre système des rôles partagés par plusieurs projets. Par exemple, vous pourriez avoir vos propres rôles installés dans votre répertoire personnel dans le sous-répertoire **~/ansible/roles**, et le système peut avoir des rôles installés pour tous les utilisateurs du répertoire **/usr/share/ansible/roles**.

Chaque rôle a son propre répertoire avec une structure de répertoire normalisée. Par exemple, la structure de répertoire suivante contient les fichiers qui définissent le rôle **motd**.

```
[user@host ~]$ tree roles/
roles/
└── motd
    ├── defaults
    │   └── main.yml
    ├── files
    ├── handlers
    └── meta
```

```

|   └── main.yml
├── README.md
└── tasks
    └── main.yml
└── templates
    └── motd.j2

```

Le fichier **README.md** fournit une description, simple et lisible par un humain, du rôle, de la documentation et des exemples d'utilisation, ainsi que toute exigence non Ansible susceptible d'être nécessaire à son fonctionnement. Le sous-répertoire **meta** contient un fichier **main.yml** qui spécifie des informations sur l'auteur, la licence, la compatibilité et les dépendances du module. Le sous-répertoire **files** contient des fichiers à contenu fixe, tandis que le sous-répertoire **templates** contient des modèles qui peuvent être déployés par le rôle lorsqu'il est utilisé. Les autres sous-répertoires peuvent contenir des fichiers **main.yml** qui définissent des gestionnaires, des tâches, des métadonnées de rôle, des variables ou encore des valeurs de variables par défaut.

Un sous-répertoire vide (comme **handlers** dans cet exemple) est ignoré. Si un rôle n'utilise pas de fonctionnalité, le sous-répertoire peut être complètement omis. Par exemple, le sous-répertoire **vars** a été omis de cet exemple.

## Création d'un squelette de rôle

Vous pouvez créer tous les sous-répertoires et fichiers nécessaires à un nouveau rôle à l'aide de commandes Linux standard. Vous pouvez également utiliser des utilitaires de ligne de commande pour automatiser le processus de création d'un nouveau rôle.

L'outil de ligne de commande **ansible-galaxy** (couvert plus en détail plus loin dans ce cours) est utilisé pour gérer les rôles Ansible, y compris la création de nouveaux rôles. Vous pouvez exécuter **ansible-galaxy init** pour créer la structure de répertoire pour un nouveau rôle. Indiquez le nom du rôle en tant qu'argument de la commande, ce qui crée un sous-répertoire pour le nouveau rôle dans le répertoire de travail en cours.

```

[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

```

## DÉFINITION DU CONTENU DU RÔLE

Une fois que vous avez créé la structure de répertoire, vous devez écrire le contenu du rôle. Un bon endroit pour commencer est le fichier de tâches **ROLENAMESPACE/tasks/main.yml**, la liste principale des tâches sont exécutées en fonction du rôle.

Le fichier **tasks/main.yml** suivant gère le fichier **/etc/motd** sur des hôtes gérés. Il utilise le module **template** pour déployer le modèle nommé **motd.j2** sur l'hôte géré. Parce que le module **template** module est configuré au sein d'une tâche de rôle, au lieu d'une tâche de playbook, le modèle **motd.j2** est récupéré à partir du sous-répertoire **templates** du rôle.

```

[user@host ~]$ cat roles/motd/tasks/main.yml
---
# tasks file for motd

- name: deliver motd file

```

```
template:
  src: motd.j2
  dest: /etc/motd
  owner: root
  group: root
  mode: 0444
```

La commande suivante affiche le contenu du modèle **motd.j2** du rôle **motd**. Elle fait référence à des faits Ansible et à une 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.
```

Le rôle définit une valeur par défaut pour la variable **system\_owner**. C'est dans le fichier **defaults/main.yml** situé dans la structure de répertoire du rôle que cette valeur est définie.

Le fichier **defaults/main.yml** suivant définit la variable **system\_owner** sur **user@host.example.com**. Il s'agira de l'adresse électronique inscrite dans le fichier **/etc/motd** des hôtes gérés auxquels ce rôle est appliqué.

```
[user@host ~]$ cat roles/motd/defaults/main.yml
---
system_owner: user@host.example.com
```

## Pratiques recommandées pour le développement de contenu de rôle

Les rôles permettent aux playbooks d'être écrits de manière modulaire. Pour optimiser l'efficacité des rôles nouvellement développés, envisagez d'appliquer les pratiques recommandées suivantes dans le développement de votre rôle :

- Conservez chaque rôle dans son propre référentiel de contrôle de version. Ansible fonctionne bien avec les dépôts basés sur **git**.
- Les informations sensibles, telles que les mots de passe ou les clés SSH, ne doivent pas être stockées dans le référentiel de rôles. Les valeurs sensibles doivent être paramétrées en tant que variables avec des valeurs par défaut non sensibles. Les playbooks qui utilisent ce rôle sont chargés de définir les variables sensibles via les fichiers de variables Ansible Vault, les variables d'environnement ou autres options **ansible-playbook**.
- Utilisez **ansible-galaxy init** pour démarrer votre rôle, puis supprimez les répertoires et les fichiers dont vous n'avez pas besoin.
- Créez et maintenez **README.md** et des fichiers **meta/main.yml** pour documenter votre rôle, qui l'a écrit et comment l'utiliser.
- Gardez votre rôle concentré sur un but ou une fonction spécifique. Au lieu de créer un rôle qui accomplit plusieurs choses, vous pouvez écrire plusieurs rôles.

- Réutilisez et refactorisez les rôles souvent. Résistez à la création de nouveaux rôles pour les configurations de périphérie. Si un rôle existant remplit la majeure partie de la configuration requise, refactorisez-le pour intégrer le nouveau scénario de configuration. Utilisez des techniques de test d'intégration et de régression pour vous assurer que le rôle fournit les nouvelles fonctionnalités requises et ne pose pas non plus de problèmes pour les playbooks existants.

## DÉFINITION DES DÉPENDANCES D'UN RÔLE

Les dépendances de rôle permettent à un rôle d'inclure d'autres rôles en tant que dépendances. Par exemple, un rôle qui définit un serveur de documentation peut dépendre d'un autre rôle qui installe et configure un serveur Web. Les dépendances sont définies dans le fichier **meta/main.yml** dans la structure de répertoire du rôle.

Voici un exemple de fichier **meta/main.yml**.

```
---
dependencies:
  - role: apache
    port: 8080
  - role: postgres
    dbname: serverlist
    admin_user: felix
```

Par défaut, les rôles ne peuvent être ajoutés en tant que dépendances à un playbook qu'une seule fois. Un rôle qui serait listé comme dépendance d'un autre rôle ne serait pas exécuté. Ce comportement peut être modifié en définissant la variable **allow\_duplicates** sur **yes** dans le fichier **meta/main.yml**.



### IMPORTANT

Limitez les dépendances de votre rôle sur d'autres rôles. Les dépendances compliquent le maintien de votre rôle, surtout s'il comporte de nombreuses dépendances complexes.

## UTILISATION DU RÔLE DANS UN PLAYBOOK

Pour accéder à un rôle, faites-y référence dans la section **roles** d'un playbook. Le playbook suivant fait référence au rôle **motd**. Étant donné qu'aucune variable n'est spécifiée, le rôle est appliqué avec ses valeurs de variable par défaut.

```
[user@host ~]$ cat use-motd-role.yml
---
- name: use motd role playbook
  hosts: remote.example.com
  user: devops
  become: true
  roles:
    - motd
```

Lors de l'exécution du playbook, les tâches effectuées conformément à un rôle peuvent être identifiées par le préfixe du nom de rôle. L'exemple de sortie suivant illustre ceci avec le préfixe **motd** : dans le nom de la tâche :

```
[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
```

Le scénario ci-dessus suppose que le rôle **motd** est situé dans le répertoire **roles**. Plus tard dans le cours, vous apprendrez à utiliser un rôle situé à distance dans un référentiel de contrôle de version.

## Modification du comportement d'un rôle avec des variables

Un rôle bien écrit utilise des variables par défaut pour modifier le comportement du rôle afin qu'il corresponde à un scénario de configuration associé. Cela contribue à rendre le rôle plus générique et réutilisable dans divers contextes.

La valeur de toute variable définie dans le répertoire **defaults** par d'un rôle sera écrasée si cette même variable est définie :

- dans un fichier d'inventaire, en tant que variable hôte ou variable de groupe ;
- dans un fichier YAML sous les répertoires **group\_vars** ou **host\_vars** d'un projet playbook ;
- comme une variable imbriquée dans le mot-clé **vars** d'un play ;
- comme variable lors de l'inclusion du rôle dans le mot-clé **roles** d'un play.

L'exemple ci-dessous montre comment utiliser le rôle **motd** avec une autre valeur pour la variable de rôle **system\_owner**. La valeur spécifiée, **someone@host.example.com**, va remplacer la référence de variable lors de l'application du rôle à un hôte géré.

```
[user@host ~]$ cat use-motd-role.yml
---
- name: use motd role playbook
  hosts: remote.example.com
  user: devops
  become: true
  vars:
    system_owner: someone@host.example.com
  roles:
    - role: motd
```

Lorsqu'elle est définie de cette manière, la variable **system\_owner** remplace la valeur de la variable par défaut du même nom. Toute définition de variable imbriquée dans le mot-clé **vars** ne remplacera pas la valeur de la même variable si elle est définie dans le répertoire **vars** d'un rôle.

L'exemple ci-dessous montre également comment utiliser le rôle **motd** avec une autre valeur pour la variable de rôle **system\_owner**. La valeur spécifiée, **someone@host.example.com**, remplacera la référence à la variable indépendamment de sa définition dans le répertoire **vars** ou **defaults** du rôle.

```
[user@host ~]$ cat use-motd-role.yml
---
- name: use motd role playbook
  hosts: remote.example.com
  user: devops
  become: true
  roles:
    - role: motd
      system_owner: someone@host.example.com
```



### IMPORTANT

La priorité des variables peut être source de confusion lorsque vous travaillez avec des variables de rôle dans un play.

- Presque toutes les autres variables remplacent les variables par défaut d'un rôle : variables d'inventaire, **vars** play, *paramètres de rôle* en ligne, etc.
- Un nombre inférieur de variables peuvent remplacer les variables définies dans le répertoire **vars** d'un rôle. Les faits, les variables chargées avec **include\_vars**, les variables enregistrées et les paramètres de rôle sont quelques-unes des variables qui peuvent le faire. Les variables d'inventaire et les **vars** de play ne le peuvent pas. Ceci est important car cela permet d'éviter que votre play ne perturbe accidentellement le fonctionnement interne du rôle.
- Toutefois, les variables déclarées en ligne en tant que paramètres de rôle, comme dans le dernier des exemples précédents, présentent une priorité très élevée. Elles peuvent remplacer les variables définies dans le répertoire **vars** d'un rôle. Si un paramètre de rôle a le même nom qu'une variable définie dans le **vars** d'un play, le **vars** d'un rôle, ou une variable d'inventaire ou de playbook, le paramètre de rôle remplace l'autre variable.



### RÉFÉRENCES

#### Utilisation de rôles – Documentation Ansible

[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks\\_reuse\\_roles.html#using-roles](https://docs.ansible.com/ansible/2.7/user_guide/playbooks_reuse_roles.html#using-roles)

#### Utilisation de variables – Documentation Ansible

[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks\\_variables.html](https://docs.ansible.com/ansible/2.7/user_guide/playbooks_variables.html)

## ► EXERCICE GUIDÉ

# CRÉATION DE RÔLES

Dans cet exercice, vous allez créer un rôle Ansible qui utilise des variables, des fichiers, des modèles, des tâches et des gestionnaires pour déployer un service réseau et activer un pare-feu opérationnel.

## RÉSULTATS

Vous devez pouvoir créer deux rôles qui utilisent des variables et des paramètres : **myvhost** et **myfirewall**.

Le rôle **myvhost** installe et configure le service Apache sur un hôte. Un modèle est fourni. Il sera utilisé pour `/etc/httpd/conf.d/vhost.conf: vhost.conf.j2`.

Le rôle **myfirewall** installe, active et démarre le daemon **firewalld**. Il ouvre le port de service du pare-feu spécifié par la variable **firewall\_service**.

À partir de `workstation`, exécutez la commande `lab role-create setup` afin de préparer l'environnement pour cet exercice. Le script crée le répertoire de travail **role-create** et l'approvisionne à l'aide d'un fichier de configuration Ansible et d'un inventaire d'hôtes.

```
[student@workstation ~]$ lab role-create setup
```

- ▶ 1. Connectez-vous à votre hôte `workstation` en tant que **student**. Accédez au répertoire de travail **role-create**.

```
[student@workstation ~]$ cd ~/role-create
[student@workstation role-create]$
```

- ▶ 2. Créez la structure de répertoire pour un rôle appelé **myvhost**. Le rôle inclut des fichiers fixes, des modèles, des tâches et des gestionnaires.

```
[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. Modifiez le fichier **main.yml** dans le sous-répertoire **tasks** du rôle. Le rôle doit effectuer quatre tâches pour s'assurer que :

- le paquetage *httpd* est installé ;
- le service *httpd* est démarré et activé ;
- le fichier de configuration du serveur Web est installé à l'aide d'un modèle fourni par le rôle ;
- le contenu HTML est téléchargé dans le répertoire **DocumentRoot** de l'hôte virtuel, comme spécifié dans le fichier de configuration.

- 3.1. Modifiez le fichier **roles/myvhost/tasks/main.yml**. Insérez un code afin d'utiliser le module **yum** pour installer le paquetage *httpd*. Le contenu du fichier doit se présenter comme suit :

```
---  
# tasks file for myvhost  
  
- name: Ensure httpd is installed  
  yum:  
    name: httpd  
    state: latest
```

- 3.2. Ajoutez du code supplémentaire au fichier **tasks/main.yml** afin d'utiliser le module **service** pour lancer et activer le service *httpd*.

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

- 3.3. Ajoutez une autre strophe afin d'utiliser le module **template** pour créer **/etc/httpd/conf.d/vhost.conf** sur l'hôte géré. Elle doit appeler un gestionnaire pour redémarrer le daemon **httpd** lorsque ce fichier est mis à jour.

```
- 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. Ajoutez une autre strophe pour copier le contenu HTML du rôle dans le répertoire **DocumentRoot** de l'hôte virtuel. Utilisez le module **copy** et insérez une barre oblique finale après le nom du répertoire source. Ce faisant, le module copie le contenu du répertoire **html** directement sous le répertoire de destination (identique

à l'utilisation de **rsync**). La variable **ansible\_hostname** se développe sur le nom court de l'hôte géré.

```
- name: HTML content is installed
copy:
  src: html/
  dest: "/var/www/vhosts/{{ ansible_hostname }}"
```

3.5. Enregistrez vos modifications, puis fermez le fichier **tasks/main.yml**.

- 4. Créez le gestionnaire pour redémarrer le service **httpd**. Modifiez le fichier **roles/myvhost/handlers/main.yml** et insérez du code pour utiliser le module **service**. Le contenu du fichier doit se présenter comme suit :

```
---
# handlers file for myvhost

- name: restart httpd
  service:
    name: httpd
    state: restarted
```

- 5. Créez le contenu HTML à diffuser par le serveur Web.

5.1. La tâche de rôle qui avait appelé le module **copy** faisait référence à un répertoire **html** en tant que **src**. Créez ce répertoire dans le sous-répertoire **files** du rôle.

```
[student@workstation role-create]$ mkdir -pv roles/myvhost/files/html
mkdir: created directory 'roles/myvhost/files/html'
```

5.2. Créez un fichier **index.html** sous ce répertoire avec le contenu suivant : « simple index ». Veillez à utiliser ce texte de chaîne en l'état, car il est recherché par le script de notation.

```
[student@workstation role-create]$ echo \
> 'simple index' > roles/myvhost/files/html/index.html
```

- 6. Déplacez le modèle **vhost.conf.j2** du répertoire de projet vers le sous-répertoire **templates** du rôle.

```
[student@workstation role-create]$ mv -v vhost.conf.j2 roles/myvhost/templates/
'vhost.conf.j2' -> 'roles/myvhost/templates/vhost.conf.j2'
```

- 7. Testez le rôle **myvhost** pour vous assurer qu'il fonctionne correctement.

7.1. Écrivez un playbook qui utilise le rôle et nommez-le **use-vhost-role.yml**. Il doit contenir les éléments suivants :

```
---
- name: Use myvhost role playbook
```

```
hosts: webservers
pre_tasks:
  - name: pre_tasks message
    debug:
      msg: 'Ensure web server configuration.'

roles:
  - myvhost

post_tasks:
  - name: post_tasks message
    debug:
      msg: 'Web server is configured.'
```

- 7.2. Avant d'exécuter le playbook, vérifiez que sa syntaxe est correcte en exécutant **ansible-playbook** avec **--syntax-check**. En cas d'erreurs, corrigez-les avant de passer à l'étape suivante. Le résultat doit ressembler à ce qui suit :

```
[student@workstation role-create]$ ansible-playbook use-vhost-role.yml \
> --syntax-check

playbook: use-vhost-role.yml
```

- 7.3. Exécutez le playbook. Examinez le résultat pour vérifier qu'Ansible a bien effectué les opérations sur le serveur Web servera.

```
[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]

TASK [myvhost : HTML content is installed] ****
changed: [servera.lab.example.com]

RUNNING HANDLER [myvhost : restart httpd] ****
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. Exécutez des commandes ad hoc pour confirmer le fonctionnement du rôle. Le paquetage `httpd` doit être installé et le service `http` doit être en cours d'exécution.

```

[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. La configuration Apache doit être installée, avec les variables du modèle développées.

```

[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. Le contenu HTML doit se trouver dans un répertoire nommé **/var/www/vhosts/servera**. Le fichier **index.html** doit contenir la chaîne « simple index ».

```

[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. Utilisez le module `uri` dans une commande ad hoc pour vérifier que le contenu Web est disponible localement. Définissez le paramètre `return_content` sur `true` pour que le contenu de la réponse du serveur soit ajouté à la sortie. Le contenu du serveur doit être la chaîne « `simple index\n` ».

```
[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. Utilisez un navigateur Web sur `workstation` pour vérifier si le contenu est disponible à partir de `http://servera.lab.example.com`. Si un pare-feu est en cours d'exécution sur `servera`, le message d'erreur suivant s'affiche :

```
[student@workstation role-create]$ curl -S http://servera.lab.example.com
curl: (7) Failed connect to servera.lab.example.com:80; No route to host
```

Cela est dû au fait que le port de pare-feu pour HTTP n'est pas ouvert. Si le contenu Web s'affiche correctement, cela signifie qu'aucun pare-feu ne s'exécute sur `servera`.

- 8. Utilisez la commande `ansible-galaxy init` pour créer la structure de répertoire pour un rôle appelé `myfirewall`.

```
[student@workstation role-create]$ cd roles/
[student@workstation roles]$ ansible-galaxy init myfirewall
- myfirewall was created successfully
[student@workstation roles]$ cd ..
[student@workstation role-create]$
```

- 9. Modifiez le fichier **main.yml** dans le sous-répertoire **tasks** du rôle **myfirewall**. Le rôle doit effectuer trois tâches :

- Installer le paquetage **vsftpd**.
- Démarrez et activez le service **firewalld**.
- Ouvrir un port de service de pare-feu.

- 9.1. Utilisez un éditeur de texte pour créer un fichier appelé **roles/myfirewall/tasks/main.yml**. Insérez un code afin d'utiliser le module **yum** pour installer le paquet **firewalld**. Le contenu du fichier doit se présenter comme suit :

```
---
# tasks file for myfirewall

- name: Ensure firewalld is installed
  yum:
    name: firewalld
    state: latest
```

- 9.2. Ajoutez du code supplémentaire au fichier **tasks/main.yml** afin d'utiliser le module **service** pour lancer et activer le service **firewalld**.

```
- name: Ensure firewalld is started and enabled
  service:
    name: firewalld
    state: started
    enabled: true
```

- 9.3. Ajoutez une autre tâche qui utilise le module **firewalld** pour ouvrir immédiatement, et de manière permanente, le port de service pour tous les services listés dans la variable **firewall\_service**. Il doit se présenter comme suit :

```
- name: Ensure firewalld services are enabled
  firewalld:
    state: enabled
    immediate: true
    permanent: true
    service: "{{ item }}"
  loop: "{{ firewall_services }}"
```

- 9.4. Enregistrez vos modifications, puis fermez le fichier **tasks/main.yml**.

- 10. Créez le fichier qui définit la valeur par défaut pour la variable **firewall\_services**. Il doit être structuré sous la forme d'une liste vide. Cela fait que le rôle, par défaut, n'ouvre aucun port.

Pour que les autres utilisateurs sachent maintenant utiliser cette variable dans leur propre playbook, fournissez un exemple de définition de la variable **firewall\_services** avec au moins une entrée. Assurez-vous que cette définition est créée sous forme de commentaires.

Lors d'une prochaine étape, vous remplacerez cette variable afin d'ouvrir le port pour le protocole HTTP lorsque vous utilisez le rôle.

Utilisez un éditeur de texte pour créer un fichier appelé **roles/myfirewall/defaults/main.yml** avec le contenu suivant :

```
---  
# defaults file for myfirewall  
  
# By default, no firewall services are enabled.  
#firewall_services: []  
#  
#To enable, for example, FTP and HTTP services in firewalld,  
# use the following definition for "firewall_services"  
#firewall_services:  
#  - http  
#  - ftp  
firewall_services: []
```

- 11. Modifiez le rôle **myvhost** pour inclure le rôle **myfirewall** en tant que dépendance, puis testez à nouveau le rôle modifié.

Utilisez un éditeur de texte pour modifier le fichier **roles/myvhost/meta/main.yml** pour rendre le rôle **myvhost** dépendant du rôle **myfirewall**. La variable **dependencies** doit être définie comme suit :

```
dependencies:  
  - role: myfirewall
```

Eventuellement, mettez à jour les champs **author**, **company**, **description** et **license** de la variable **galaxy\_info**. Enregistrez le fichier **roles/myvhost/meta/main.yml**.

- 12. Créez un fichier YAML sous le répertoire **group\_vars** pour remplacer la variable **firewall\_services** pour tous les hôtes **webserver**.

La variable **firewall\_services** doit contenir un service, **ssh**. Les ports pour les protocoles HTTP seront ouverts par le rôle **myfirewall**.

- 12.1. Créez un répertoire **group\_vars/webservers** pour contenir des fichiers YAML qui définissent des variables pour le groupe d'hôtes **webservers**.

```
[student@workstation role-create]$ mkdir -pv group_vars/webservers  
mkdir: created directory 'group_vars'
```

```
mkdir: created directory 'group_vars/webservers'
```

- 12.2. Copiez le fichier **defaults/main.yml** pour le rôle **myfirewall** dans le répertoire **group\_vars/webservers**. Nommez le fichier d'après le rôle, **myfirewall.yml**.

```
[student@workstation role-create]$ cp -v \
> roles/myfirewall/defaults/main.yml group_vars/webservers/myfirewall.yml
'roles/myfirewall/defaults/main.yml' -> 'group_vars/webservers/myfirewall.yml'
```

- 12.3. Mettez à jour le fichier **group\_vars/webservers/myfirewall.yml** avec les valeurs souhaitées pour la variable **firewall\_services**. Supprimez également le commentaire **#** du fichier. Le contenu du fichier obtenu doit se présenter comme suit :

```
---
firewall_services:
  - http
```

- 13. Exécutez de nouveau le playbook. Confirmez l'exécution correcte des tâches **myfirewall** supplémentaires.

```
[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 [myfirewall : Ensure firewalld is installed] ****
ok: [servera.lab.example.com]

TASK [myfirewall : Ensure firewalld is started and enabled] ****
ok: [servera.lab.example.com]

TASK [myfirewall : Ensure firewalld services are enabled] ****
changed: [servera.lab.example.com] => (item=http)

TASK [myvhost : Ensure httpd is installed] ****
ok: [servera.lab.example.com]

TASK [myvhost : Ensure httpd is started and enabled] ****
ok: [servera.lab.example.com]

TASK [myvhost : vhost file is installed] ****
ok: [servera.lab.example.com]
```

```
TASK [myvhost : HTML content is installed] ****
ok: [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=10    changed=1    unreachable=0    failed=0
```

- 14. Confirmez la disponibilité du contenu du serveur Web pour les clients distants.

```
[student@workstation role-create]$ curl http://servera.lab.example.com
simple index
```

## Nettoyage

Exécutez la commande **lab role-create cleanup** pour nettoyer l'hôte géré.

```
[student@workstation ~]$ lab role-create cleanup
```

L'exercice guidé est maintenant terminé.

# DÉPLOIEMENT DE RÔLES AVEC ANSIBLE GALAXY

---

## OBJECTIFS

Au terme de cette section, les stagiaires doivent pouvoir sélectionner et récupérer des rôles à partir d'Ansible Galaxy ou d'autres sources, telles qu'un référentiel Git, et les utiliser dans des playbooks.

## ANSIBLE GALAXY

Ansible Galaxy [<https://galaxy.ansible.com>] est une bibliothèque publique de contenu Ansible, élaboré par une communauté d'administrateurs et d'utilisateurs Ansible. Elle contient des milliers de rôles Ansible et intègre une base de données interrogeable pour aider les utilisateurs d'Ansible à identifier les rôles susceptibles de les aider à effectuer une tâche administrative précise. Ansible Galaxy comprend des liens vers de la documentation et des vidéos à l'intention des nouveaux utilisateurs et des développeurs de rôles Ansible.

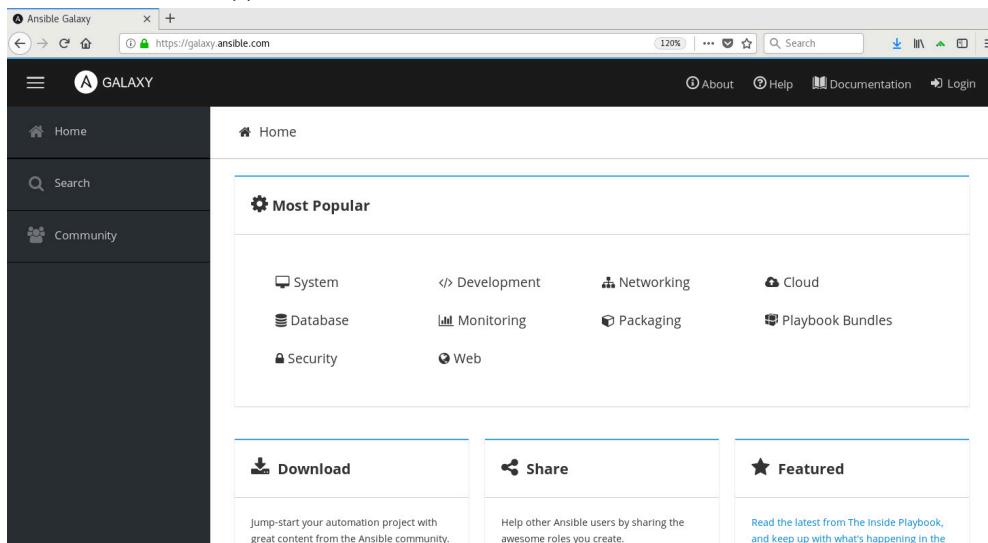


Figure 8.1: Page d'accueil Ansible Galaxy

De plus, la commande **ansible-galaxy** que vous utilisez pour obtenir et gérer les rôles d'Ansible Galaxy peut également être utilisée pour obtenir et gérer les rôles dont vos projets ont besoin à partir de vos propres référentiels Git.

## Aide sur Ansible Galaxy

L'onglet Documentation disponible sur la page d'accueil du site Web Ansible Galaxy pointe vers une page qui décrit l'utilisation d'Ansible Galaxy. Vous y apprendrez notamment comment télécharger et utiliser des rôles à partir d'Ansible Galaxy. Cette page contient également des instructions relatives au développement de rôles et à leur téléchargement sur Ansible Galaxy.

## Rechercher les rôles dans Ansible Galaxy

L'onglet Search sur le côté gauche de la page d'accueil du site Web Ansible Galaxy permet aux utilisateurs d'accéder à des informations relatives aux rôles publiés sur Ansible Galaxy. Vous

pouvez rechercher un rôle Ansible en fonction de son nom, en utilisant des balises ou d'autres attributs de rôle. Les résultats sont présentés par ordre décroissant selon le score **Best Match** (Meilleure correspondance), qui est un score calculé en fonction de la qualité du rôle, de la popularité du rôle et de critères de recherche.



### NOTE

Évaluation du contenu [[https://galaxy.ansible.com/docs/contributing/content\\_scoring.html](https://galaxy.ansible.com/docs/contributing/content_scoring.html)] dans la documentation offre des informations supplémentaires sur la façon dont les rôles sont évalués par Ansible Galaxy.

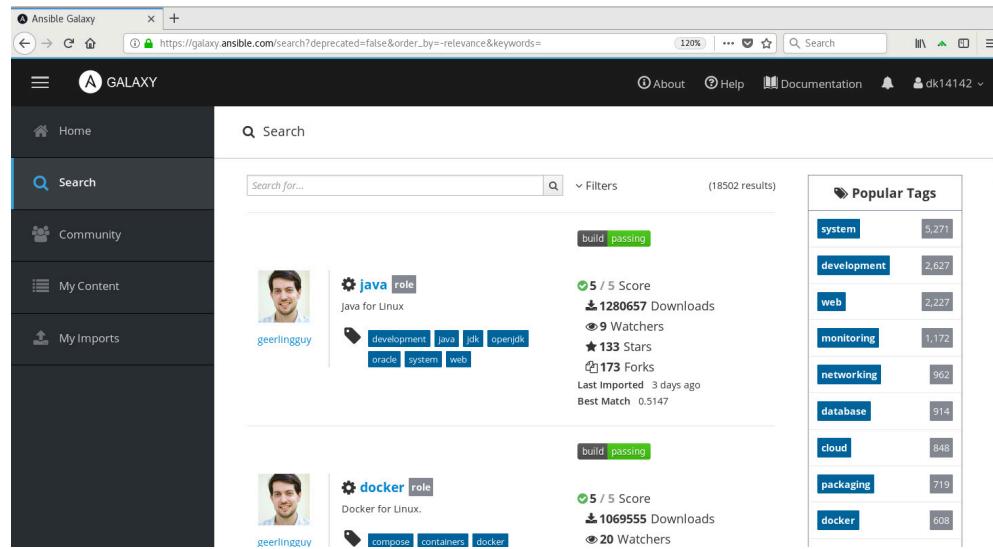


Figure 8.2: Écran de recherche d'Ansible Galaxy

Ansible Galaxy indique le nombre de fois que chaque rôle a été téléchargé à partir d'Ansible Galaxy. En outre, Ansible Galaxy indique également le nombre d'observateurs, de forks et d'étoiles du référentiel GitHub du rôle. Les utilisateurs peuvent utiliser ces informations pour déterminer le niveau de développement actif d'un rôle et sa popularité dans la communauté.

La capture d'écran ci-dessous illustre les résultats de recherche affichés par Ansible Galaxy après l'exécution d'une recherche par mot-clé sur **redis**. Notez que le premier résultat a un score **Best Match de 0, 9009**.

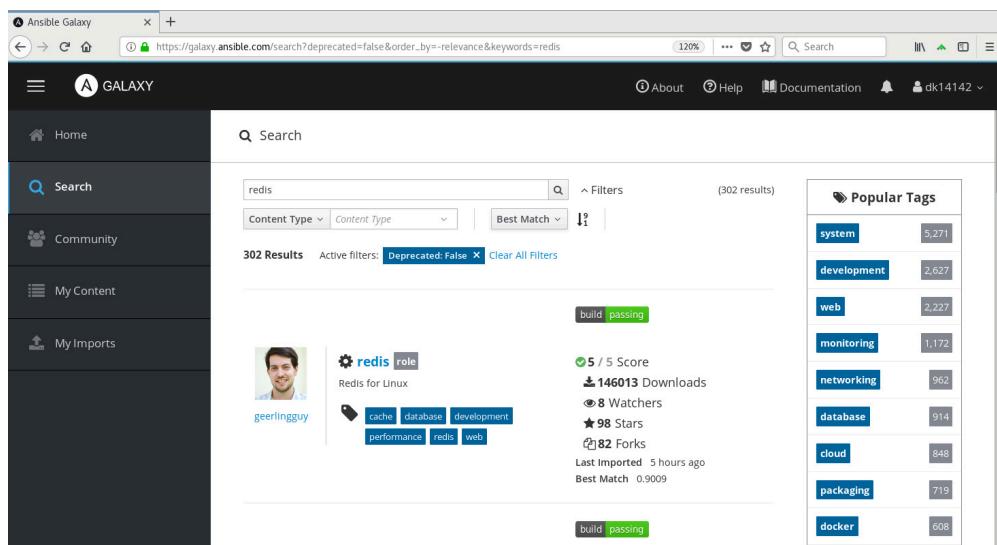


Figure 8.3: Exemple de résultats de recherche Ansible Galaxy

Le menu déroulant **Filters** situé à droite de la zone de recherche permet d'effectuer des recherches sur des mots-clés, des ID d'auteur, des plateformes et des balises. Les valeurs de plateforme possibles incluent **EL** pour Red Hat Enterprise Linux (et les distributions associées proches telles que CentOS) et **Fedora**, entre autres.

Les balises sont des chaînes d'un seul mot arbitraires et définies par l'auteur du rôle qui décrivent et classent le rôle. Les utilisateurs peuvent utiliser des balises pour trouver des rôles pertinents. Les valeurs possibles pour les balises sont **system**, **development**, **web**, **monitoring**, etc. Un rôle peut avoir jusqu'à 20 balises dans Ansible Galaxy.



#### IMPORTANT

Dans l'interface de recherche Ansible Galaxy, les recherches par mot-clé correspondent à des mots ou des expressions dans le fichier **README**, au nom du contenu ou à la description du contenu. Les recherches de balises, en revanche, correspondent spécifiquement aux valeurs de balise définies par l'auteur pour le rôle.

## OUTIL DE LIGNE DE COMMANDE ANSIBLE GALAXY

Vous pouvez utiliser l'outil de ligne de commande **ansible-galaxy** pour rechercher, installer, lister, supprimer ou initialiser des rôles, ou encore afficher des informations à leur sujet.

### Recherche de rôles à partir de la ligne de commande

La sous-commande **ansible-galaxy search** recherche des rôles dans Ansible Galaxy. Si vous spécifiez une chaîne en tant qu'argument, elle est utilisée pour rechercher des rôles par mot-clé dans Ansible Galaxy. Vous pouvez utiliser les options **--author**, **--platforms** et **--galaxy-tags** pour limiter les résultats de la recherche. Vous pouvez également utiliser ces options comme clé de recherche principale. Par exemple, la commande **ansible-galaxy search --author geerlingguy** affichera tous les rôles soumis par l'utilisateur geerlingguy.

Les résultats sont affichés par ordre alphabétique, pas par score **Best Match** dans l'ordre décroissant. L'exemple suivant affiche les noms des rôles dont la description contient **redis** et qui sont disponibles pour la plateforme Enterprise Linux (**EL**).

```
[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...
  gearlingguy.php-redis  PhpRedis support for Linux
  gearlingguy.redis  Redis for Linux
  gikoluo.filebeat  Filebeat for Linux.
  ...output omitted...
```

La sous-commande **ansible-galaxy info** affiche des informations plus détaillées sur un rôle. Ansible Galaxy obtient cette information de plusieurs endroits, y compris le fichier **meta/main.yml** du rôle et son référentiel GitHub. La commande suivante affiche des informations sur le rôle **gearlingguy.redis**, disponible auprès d'Ansible Galaxy.

```
[user@host ~]$ ansible-galaxy info gearlingguy.redis

Role: gearlingguy.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: gearlingguy
  ...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...
```

## Installation de rôles depuis Ansible Galaxy

La sous-commande **ansible-galaxy install** télécharge un rôle depuis Ansible Galaxy, puis l'installe en local sur le nœud de contrôle.

Par défaut, les rôles sont installés dans le premier répertoire accessible en écriture dans le **roles\_path** de l'utilisateur. En fonction du **roles\_path** par défaut défini pour Ansible, le rôle sera normalement installé dans le répertoire **~/ansible/roles** de l'utilisateur. Le **roles\_path** par défaut peut être remplacé par votre fichier de configuration Ansible actuel

ou par la variable d'environnement `ANSIBLE_ROLES_PATH`, qui affecte le comportement de **ansible-galaxy**.

Vous pouvez également spécifier un répertoire spécifique dans lequel installer le rôle à l'aide de l'option `-p DIRECTORY`.

Dans l'exemple suivant, **ansible-galaxy** installe le rôle `geerlingguy.redis` dans le répertoire **roles** du projet de playbook. Le répertoire de travail courant de la commande est `/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 ~]$ ls roles/
geerlingguy.redis
```

## Installation de rôles à l'aide d'un fichier de configuration requise

Vous pouvez aussi utiliser **ansible-galaxy** pour installer une liste de rôles basée sur des définitions dans un fichier texte. Par exemple, si vous avez un playbook dans lequel des rôles spécifiques doivent être installés, vous pouvez créer un fichier **roles/requirements.yml** dans le répertoire du projet qui spécifie les rôles nécessaires. Ce fichier sert de manifeste de dépendance pour le projet de playbook, ce qui permet de développer et de tester les playbooks séparément de tout rôle de support.

Par exemple, un **requirements.yml** simple pour installer `geerlingguy.redis` peut se présenter comme ceci :

```
- src: geerlingguy.redis
  version: "1.5.0"
```

L'attribut `src` spécifie la source du rôle, en l'occurrence le rôle `geerlingguy.redis` d'Ansible Galaxy. L'attribut `version` est facultatif et spécifie la version du rôle à installer, en l'occurrence **1.5.0**.



### IMPORTANT

Vous devez spécifier la version du rôle dans votre fichier **requirements.yml**, en particulier pour les playbooks en production.

Si vous ne spécifiez pas de version, vous obtiendrez la dernière version du rôle.

Si l'auteur en amont apporte des modifications au rôle incompatibles avec votre playbook, cela peut entraîner une défaillance de l'automatisation ou d'autres problèmes.

Pour installer les rôles à l'aide d'un fichier de rôle, utilisez l'option `-r REQUIREMENTS-FILE` :

```
[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
```

Vous pouvez utiliser **ansible-galaxy** installer des rôles ne figurant pas dans Ansible Galaxy. Vous pouvez héberger vos propres rôles internes ou propriétaires dans un référentiel Git privé ou sur un serveur Web. L'exemple suivant montre comment configurer un fichier de configuration à l'aide de diverses sources distantes.

```
[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
```

Le mot-clé **src** spécifie le nom du rôle Ansible Galaxy. Si le rôle n'est pas hébergé sur Ansible Galaxy, le mot-clé **src** clé indique l'URL du rôle.

Si le rôle est hébergé dans un référentiel de contrôle de source, l'attribut **scm** est requis. La commande **ansible-galaxy** est capable de télécharger et d'installer des rôles depuis un référentiel logiciel basé sur Git ou Mercurial. Un référentiel basé sur Git nécessite une valeur **scm** correspondant à **git**, alors qu'un rôle hébergé sur un référentiel Mercurial requiert une valeur **hg**. Si le rôle est hébergé sur Ansible Galaxy ou en tant qu'archive tar sur un serveur Web, le mot-clé **scm** est omis.

Le mot-clé **name** est utilisé pour remplacer le nom local du rôle. Le mot-clé **version** est utilisé pour spécifier la version d'un rôle. Le mot-clé **version** peut être toute valeur correspondant à une branche, une balise ou un hachage de validation du référentiel logiciel du rôle.

Pour installer les rôles associés à un projet de playbook, exécutez la commande **ansible-galaxy install**:

```
[user@host project]$ ansible-galaxy install -r roles/requirements.yml \
```

```
> -p roles
- downloading role 'redis', owned by gearlingguy
- downloading role from https://github.com/gearlingguy/ansible-role-redis/
archive/1.6.0.tar.gz
- extracting gearlingguy.redis to /opt/project/roles/gearlingguy.redis
- gearlingguy.redis (1.6.0) was installed successfully
- downloading role 'redis', owned by gearlingguy
- downloading role from https://github.com/gearlingguy/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
```

## Gestion des rôles téléchargés

La commande **ansible-galaxy** peut également gérer des rôles locaux, tels que ceux trouvés dans le répertoire **roles** d'un projet de playbook. La sous-commande **ansible-galaxy list** liste les rôles disponibles en local.

```
[user@host project]$ ansible-galaxy list
- gearlingguy.redis, 1.6.0
- myrole, (unknown version)
- nginx-acme, 56e00a54
- nginx-acme-ssh, master
- redis_prod, 1.5.0
```

Il est possible de supprimer un rôle en local à l'aide de la sous-commande **ansible-galaxy remove**.

```
[user@host ~]$ ansible-galaxy remove nginx-acme-ssh
- successfully removed nginx-acme-ssh
[user@host ~]$ ansible-galaxy list
- gearlingguy.redis, 1.6.0
- myrole, (unknown version)
- nginx-acme, 56e00a54
- redis_prod, 1.5.0
```

Utilisez les rôles téléchargés et installés dans des playbooks comme n'importe quel autre rôle. Ils peuvent être référencés dans la section **roles** à l'aide de leur nom de rôle téléchargé. Si un rôle ne figure pas dans le répertoire **roles** du projet, le **roles\_path** sera vérifié pour voir si le rôle est installé dans l'un de ces répertoires, la première correspondance étant utilisée. Le **use-role.yml** playbook suivant référence les rôles **redis\_prod** et **gearlingguy.redis**:

```
[user@host project]$ cat use-role.yml
---
- name: use redis_prod for Prod machines
```

```
hosts: redis_prod_servers
user: devops
become: true
roles:
  - redis_prod

- name: use geerlingguy.redis for Dev machines
hosts: redis_dev_servers
user: devops
become: true
roles:
  - geerlingguy.redis
```

Ce playbook entraîne l'application de différentes versions du rôle `geerlingguy.redis` aux serveurs de production et de développement. De cette manière, les modifications apportées au rôle peuvent être systématiquement testées et intégrées avant leur déploiement sur les serveurs de production. Si une modification récente d'un rôle cause des problèmes, l'utilisation du contrôle de version pour développer le rôle vous permet de revenir à une version précédente et stable du rôle.



## RÉFÉRENCES

### Ansible Galaxy – Documentation Ansible

[https://docs.ansible.com/ansible/2.7/reference\\_appendices/galaxy.html](https://docs.ansible.com/ansible/2.7/reference_appendices/galaxy.html)

## ► EXERCICE GUIDÉ

# DÉPLOIEMENT DE RÔLES AVEC ANSIBLE GALAXY

Au cours de cet exercice, vous allez utiliser Ansible Galaxy pour télécharger et installer un rôle Ansible.

## RÉSULTATS

Vous devez pouvoir réaliser les tâches suivantes :

- créer un fichier de rôles pour spécifier les dépendances de rôle d'un playbook ;
- installer les rôles spécifiés dans un fichier de rôles ;
- lister des rôles en utilisant la commande **ansible-galaxy**.

## PRÉSENTATION DU SCÉNARIO

Votre organisation place des fichiers personnalisés dans le répertoire **/etc/skel** sur tous les hôtes. En conséquence, les nouveaux comptes d'utilisateur sont configurés avec un environnement Bash normalisé spécifique à l'organisation.

Vous allez tester la version de développement du rôle Ansible responsable du déploiement des fichiers squelette de l'environnement Bash.

À partir de **workstation**, exécutez la commande **lab role-galaxy setup** afin de préparer l'environnement pour cet exercice. Le script crée le répertoire de travail **role-galaxy** et l'approvisionne à l'aide d'un fichier de configuration Ansible et d'un inventaire d'hôtes.

```
[student@workstation ~]$ lab role-galaxy setup
```

- 1. Connectez-vous à votre hôte **workstation** en tant que **student**. Accédez au répertoire de travail **role-galaxy**.

```
[student@workstation ~]$ cd ~/role-galaxy
[student@workstation role-galaxy]$
```

- 2. Pour tester le rôle Ansible qui configure les fichiers du squelette, ajoutez la spécification du rôle à un fichier de rôles.

Lancez votre éditeur de texte favori et créez un fichier appelé **requirements.yml** dans le sous-répertoire **roles**. L'URL du référentiel Git du rôle est : `git@workstation.lab.example.com:student/bash_env`. Pour voir comment le rôle affecte le comportement des hôtes de production, utilisez la branche **master** du référentiel. Définissez le nom local du rôle sur **student.bash\_env**.

Le fichier **roles/requirements.yml** contient à présente le contenu suivant :

```
---
```

```
# requirements.yml

- src: git@workstation.lab.example.com:student/bash_env
  scm: git
  version: master
  name: student.bash_env
```

- 3. Exécutez la commande **ansible-galaxy** pour utiliser le fichier de rôles que vous venez de créer et installez le rôle **student.bash\_env**.

- 3.1. Affichez le contenu du sous-répertoire **roles** avant l'installation du rôle, à des fins de comparaison.

```
[student@workstation role-galaxy]$ ls roles/
requirements.yml
```

- 3.2. Utilisez Ansible Galaxy pour télécharger et installer le rôle listé dans le fichier **roles/requirements.yml**. Assurez-vous que tous les rôles téléchargés sont stockés dans le sous-répertoire **roles**.

```
[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. Affichez le sous-répertoire **roles** après l'installation du rôle. Confirmez qu'il contient un nouveau sous-répertoire nommé **student.bash\_env**, correspondant à la valeur **name** spécifiée dans le fichier YAML.

```
[student@workstation role-galaxy]$ ls roles/
requirements.yml  student.bash_env
```

- 3.4. Essayez d'utiliser la commande **ansible-galaxy**, sans aucune option, pour lister les rôles du projet :

```
[student@workstation role-galaxy]$ ansible-galaxy list
[WARNING]: - the configured path /home/student/.ansible/roles does not exist.
```

Parce que vous avez utilisé l'option **-p** avec la commande **ansible-galaxy install**, le rôle **student.bash\_env** n'a pas été installé à l'emplacement par défaut. Utilisez l'option **-p** avec la commande **ansible-galaxy list** pour lister les rôles téléchargés :

```
[student@workstation role-galaxy]$ ansible-galaxy list -p roles
- student.bash_env, master
```

```
[WARNING]: - the configured path /home/student/.ansible/roles does not exist.
```



### NOTE

Le répertoire **/home/student/.ansible/roles** se trouve dans votre **roles\_path** par défaut, mais puisque vous n'avez pas tenté d'installer un rôle sans utiliser l'option **-p, ansible-galaxy** n'a pas encore créé le répertoire.

- ▶ 4. Créez le playbook **use-bash\_env-role.yml** qui utilise le rôle **student.bash\_env**. Le contenu du playbook doit correspondre à ce qui suit :

```
---
- 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') }}"
```

Pour voir les effets du changement de configuration, un nouveau compte utilisateur doit être créé. La section **pre\_tasks** et **post\_tasks** du playbook permet de s'assurer que le compte utilisateur **student2** est créé à chaque exécution du playbook. Après l'exécution du playbook, le compte **student2** est accessible avec le mot de passe **redhat**.

- ▶ 5. Exécutez le playbook. Le rôle **student.bash\_env** crée des fichiers de configuration de modèle standard dans **/etc/skel** sur l'hôte géré. Les fichiers créés comprennent **.bashrc**, **.bash\_profile** et **.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=3      unreachable=0      failed=0

```

- ▶ 6. Connotez-vous à servera comme utilisateur student2 en utilisant SSH. Observez l'invite personnalisée de l'utilisateur student2, puis déconnectez-vous de servera.

```

[student@workstation role-galaxy]$ ssh student2@servera
[student2 on servera in ~ dir]$ exit
logout
Connection to servera closed.
[student@workstation role-galaxy]$

```

- ▶ 7. Exécutez le playbook en utilisant la version de développement du rôle student.bash\_env.
- La version de développement du rôle est située dans la branche **dev** du référentiel Git. La version de développement du rôle utilise une nouvelle variable, **prompt\_color**. Avant d'exécuter le playbook, ajoutez la variable **prompt\_color** à la section **vars** du playbook et définissez sa valeur sur**blue**.
- 7.1. Mettez à jour le fichier **roles/requirements.yml** et définissez la valeur **version** sur **dev**. Le fichier **roles/requirements.yml** contient désormais :

```

---
# requirements.yml

- src: git@workstation.lab.example.com:student/bash_env
  scm: git
  version: dev
  name: student.bash_env

```

- 7.2. Utilisez la commande **ansible-galaxy install** pour installer le rôle à l'aide du fichier de rôles mis à jour. Utilisez l'option **--force** pour remplacer la version **master** du rôle par la version **dev** du rôle.

```

[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. Modifiez le fichier **use-bash\_env-role.yml**. Ajoutez la variable **prompt\_color** avec la valeur **blue** à la section **vars** du playbook. Le fichier contient désormais :

```
---
- name: use student.bash_env role playbook
  hosts: devservers
  vars:
    prompt_color: blue
    default_prompt: '[\u on \h in \W]\$ '
  pre_tasks:
...output omitted...
```

- 7.4. Exécutez le playbook **use-bash\_env-role.yml**.

```
[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] ****
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. Connctez-vous à nouveau à **servera** comme utilisateur **student2** en utilisant SSH. Observez l'erreur de l'utilisateur **student2**, puis déconnectez-vous de **servera**.

```
[student@workstation role-galaxy]$ ssh student2@servera
-bash: [: missing `]'
-bash-4.2$ exit
logout
Connection to servera closed.
```

```
[student@workstation role-galaxy]$
```

Une erreur Bash s'est produite lors de l'analyse du fichier **.bash\_profile** de l'utilisateur student2.

- 9. Corrigez l'erreur dans la version de développement du rôle `student.bash_env`, puis réexécutez le playbook.
- 9.1. Modifiez le fichier **roles/student.bash\_env/templates/.bash\_profile.j2**. Ajoutez le caractère `]` manquant à la ligne 4 et enregistrez le fichier. La partie supérieure du fichier se présente maintenant comme suit :

```
# .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
```

Enregistrez le fichier.

- 9.2. Exécutez le playbook **use-bash\_env-role.yml**.

```
[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. Connectez-vous à nouveau à servera comme utilisateur student2 en utilisant SSH.

```
[student@workstation role-galaxy]$ ssh student2@servera
[student2 on servera in ~ dir]$ exit
logout
Connection to servera closed.
[student@workstation role-galaxy]$
```

Le message d'erreur n'est plus présent. L'invite personnalisée pour l'utilisateur student2 s'affiche maintenant avec des caractères bleus.

Les étapes ci-dessus démontrent que la version de développement du rôle `student .bash_env` est défectueuse. Sur la base des résultats des tests, les développeurs vont valider les correctifs nécessaires dans la branche de développement du rôle. Lorsque la branche de développement passe les contrôles de qualité requis, les développeurs fusionnent les fonctionnalités de la branche de développement avec la branche `master`.

La validation des changements de rôle dans un référentiel Git dépasse bien entendu le cadre de ce cours.



### IMPORTANT

Si vous suivez la dernière version d'un rôle dans votre projet, réinstallez-le régulièrement pour le mettre à jour. Cela garantit que votre copie locale reste à jour avec des correctifs de bogues, des correctifs et d'autres fonctionnalités.

Par contre, si vous utilisez un rôle tiers en production, vous devez spécifier la version que vous souhaitez utiliser afin d'éviter les interruptions dues à des modifications inattendues. Dans ce cas, vous devez mettre à jour régulièrement la dernière version du rôle dans votre environnement de test afin que vous puissiez adopter les améliorations et les modifications de manière contrôlée.

## Nettoyage

Exécutez la commande `lab role-galaxy cleanup` pour nettoyer l'hôte géré.

```
[student@workstation ~]$ lab role-galaxy cleanup
```

L'exercice guidé est maintenant terminé.

# RÉUTILISATION DU CONTENU AVEC DES RÔLES SYSTÈME

---

## OBJECTIFS

Après avoir terminé cette section, les stagiaires doivent pouvoir écrire des playbooks qui tirent parti des rôles système Red Hat Enterprise Linux pour effectuer des opérations standard.

## RÔLES SYSTÈME RED HAT ENTERPRISE LINUX

À partir de Red Hat Enterprise Linux 7.4, un certain nombre de rôles Ansible ont été fournis avec le système d'exploitation dans le paquetage *rhel-system-roles* via le canal de distribution Extras. Voici une brève description de chaque rôle :

### Rôles système RHEL

| NOM                                     | ÉTAT                       | DESCRIPTION DU RÔLE                                                                                                                                                 |
|-----------------------------------------|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>rhel-system-roles.kdump</code>    | Entièrement pris en charge | Configure le service kdump de récupération après incident.                                                                                                          |
| <code>rhel-system-roles.network</code>  | Entièrement pris en charge | Configure les interfaces réseau.                                                                                                                                    |
| <code>rhel-system-roles.selinux</code>  | Entièrement pris en charge | Configure et gère la personnalisation SELinux, y compris le mode SELinux, les contextes de fichier et de port, les paramètres booléens et les utilisateurs SELinux. |
| <code>rhel-system-roles.timesync</code> | Entièrement pris en charge | Configure la synchronisation de l'horloge à l'aide du protocole NTP (Network Time Protocol) ou du protocole PTP (Precision Time Protocol).                          |
| <code>rhel-system-roles.postfix</code>  | Préversion technologique   | Configure chaque hôte en tant qu'agent MTA (Mail Transfer Agent) à l'aide du service Postfix.                                                                       |
| <code>rhel-system-roles.firewall</code> | En cours de développement  | Configure le pare-feu d'un hôte.                                                                                                                                    |
| <code>rhel-system-roles.tuned</code>    | En cours de développement  | Configure le service tuned pour ajuster les performances du système.                                                                                                |

Les rôles système visent à normaliser la configuration des sous-systèmes Red Hat Enterprise Linux, sur plusieurs versions de Red Hat Enterprise Linux. Utilisez les rôles système pour configurer n'importe quel hôte Red Hat Enterprise Linux, version 6.10 et ultérieure.

## Gestion de configuration simplifiée

Par exemple, le service de synchronisation de l'horloge recommandé pour Red Hat Enterprise Linux 7 est le service `chrony`. Cependant, dans Red Hat Enterprise Linux 6, le service recommandé est `ntpd`. Dans un environnement combinant des hôtes Red Hat Enterprise Linux 6 et 7, un administrateur doit gérer les fichiers de configuration des deux services.

Avec les rôles système RHEL, les administrateurs n'ont plus besoin de gérer les fichiers de configuration des deux services. Les administrateurs peuvent utiliser le rôle `rhel-system-roles.timesync` pour configurer la synchronisation de l'horloge des hôtes Red Hat Enterprise Linux 6 et 7. Un fichier YAML simplifié contenant des variables de rôle définit la configuration de la synchronisation de l'horloge pour les deux types d'hôtes.

## Prise en charge des rôles système RHEL

Les rôles système RHEL sont issus du projet Open Source Linux System Roles, disponible sur Ansible Galaxy. Contrairement aux rôles système Linux, les rôles système RHEL sont pris en charge par Red Hat dans le cadre d'un abonnement Red Hat Enterprise Linux standard. Les rôles système RHEL bénéficient des mêmes avantages en termes de support de cycle de vie qu'un abonnement Red Hat Enterprise Linux.

Chaque rôle système est testé et stable. Les rôles système **entièvement pris en charge** ont également des interfaces stables. Tous les rôles système **entièvement pris en charge** continueront à utiliser les mêmes variables de rôle dans les versions futures. La refactorisation des playbooks liée aux changements de rôle système devrait être minime.

Les rôles système de **préversion technologique** pourront utiliser différentes variables de rôle dans les versions futures. Un test d'intégration est recommandé pour les playbooks incorporant tout rôle de **préversion technologique**. Les playbooks peuvent nécessiter une refactorisation si les variables de rôle changent dans une version ultérieure du rôle.

D'autres rôles sont en cours de développement dans le projet Linux System Roles en amont, mais ne sont pas encore disponibles dans le cadre d'un abonnement RHEL. Ces rôles sont disponibles via Ansible Galaxy.

## INSTALLATION DES RÔLES SYSTÈME RHEL

Les rôles système RHEL sont fournis par le paquetage `rhel-system-roles`, disponible via le canal de distribution RHEL Extras. Installez ce paquetage sur le nœud de contrôle Ansible.

Utilisez la procédure suivante pour installer le paquetage `rhel-system-roles`. La procédure suppose que le nœud de contrôle est enregistré dans un abonnement Red Hat Enterprise Linux et qu'Ansible Engine est installé. Reportez-vous à la section relative à *Installation d'Ansible* pour plus d'informations.

1. Activez le canal de distribution Extras.

```
[root@host ~]# subscription-manager repos --enable rhel-7-server-extras-rpms
```

2. Installez les rôles système RHEL.

```
[root@host ~]# yum install rhel-system-roles
```

Après l'installation, les rôles système RHEL sont situés dans le répertoire `/usr/share/ansible/roles`:

```
[root@host ~]# ls -l /usr/share/ansible/roles/
total 20
...output omitted... linux-system-roles.kdump -> rhel-system-roles.kdump
...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
```

Le nom en amont correspondant à chaque rôle est lié au rôle système RHEL. Cela permet à un rôle d'être référencé dans un playbook par l'un ou l'autre nom.

Le chemin d'accès `roles_path` par défaut sur Red Hat Enterprise Linux comprend `/usr/share/ansible/roles`. Ansible doit donc rechercher automatiquement ces rôles lorsqu'ils sont référencés par le playbook.



#### NOTE

Il se peut qu'Ansible ne trouve pas les rôles système si `roles_path` a été remplacé dans le fichier de configuration Ansible actuel, si la variable d'environnement `ANSIBLE_ROLES_PATH` est définie, ou s'il existe un autre rôle du même nom dans un répertoire listé précédemment dans `roles_path`.

## Accès à la documentation relative aux rôles système RHEL

Après l'installation, la documentation relative aux rôles système RHEL est située dans le répertoire `/usr/share/doc/rhel-system-roles-<version>/`. La documentation est organisée en sous-répertoires par sous-système :

```
[root@host ~]# ls -l /usr/share/doc/rhel-system-roles-1.0/
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
```

Le répertoire de documentation de chaque rôle contient un fichier **README.md**. Le fichier **README.md** contient une description du rôle, ainsi que des informations sur son utilisation.

Le fichier **README.md** décrit également les variables de rôle qui affectent le comportement du rôle. Le fichier **README.md** contient souvent un extrait de playbook illustrant les paramètres de variable d'un scénario de configuration courant.

Certains répertoires de documentation sur les rôles contiennent des exemples de playbooks. Lorsque vous utilisez un rôle pour la première fois, examinez tous les exemples de playbooks supplémentaires dans le répertoire de documentation.

La documentation sur les rôles concernant les rôles système RHEL correspond à celle afférente aux rôles système Linux. Utilisez un navigateur Web pour accéder à la documentation sur les rôles en amont sur le site Ansible Galaxy, <https://galaxy.ansible.com>.

## EXEMPLE DU RÔLE DE SYNCHRONISATION DE L'HORLOGE

Supposons que vous deviez configurer la synchronisation de l'horloge NTP sur vos serveurs. Vous pouvez écrire vous-même le script d'automatisation pour effectuer chacune des tâches nécessaires. Mais les rôles système RHEL incluent un rôle qui peut s'en charger, `rhel-system-roles.timesync`.

Le rôle est documenté dans le fichier **README.md** du répertoire **/usr/share/doc/rhel-system-roles-1.0/timesync**. Le fichier décrit toutes les variables qui affectent le comportement du rôle et contient trois extraits de playbooks illustrant différentes configurations de synchronisation de l'horloge.

Pour configurer manuellement les serveurs NTP, le rôle a une variable nommée `timesync_ntp_servers`. Elle prend une liste de serveurs NTP à utiliser. Chaque élément de la liste est composé d'un ou de plusieurs attributs. Les deux attributs clés sont :

### Attributs `timesync_ntp_servers`

| ATTRIBUT              | OBJET                                                                                                                                                                              |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>hostname</code> | Nom d'hôte d'un serveur NTP avec lequel effectuer la synchronisation.                                                                                                              |
| <code>iburst</code>   | Attribut booléen qui active ou désactive la synchronisation initiale rapide. Sa valeur par défaut étant <b>no</b> dans le rôle, vous devez normalement le définir sur <b>yes</b> . |

Étant donné ces informations, l'exemple suivant est un play qui utilise le rôle `rhel-system-roles.timesync` pour configurer les hôtes gérés afin d'obtenir l'heure de trois serveurs NTP à l'aide de la synchronisation initiale rapide. De plus, une tâche utilisant le module `timezone` a été ajoutée pour définir le fuseau horaire des hôtes sur UTC.

```
- 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 }}"
```

**NOTE**

Si vous souhaitez définir un fuseau horaire différent, vous pouvez utiliser la commande **tzselect** pour rechercher d'autres valeurs valides. Vous pouvez également utiliser la commande **timedatectl** pour vérifier les réglages actuels de l'horloge.

Cet exemple définit les variables de rôle dans une section **vars** du play, mais une meilleure pratique consisterait à les configurer en tant que variables d'inventaire pour des hôtes ou des groupes d'hôtes.

Considérons un projet de playbook avec la structure suivante :

```
[root@host playbook-project]# tree
.
├── ansible.cfg
├── group_vars
│   └── servers
│       └── timesync.yml①
└── inventory
└── timesync_playbook.yml②
```

- ① Définit les variables de synchronisation de l'horloge en remplaçant les rôles par défaut pour les hôtes du groupe **servers** dans l'inventaire. Ce fichier ressemblerait à ce qui suit :

```
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 Le contenu du playbook est simplifié comme suit :

```
- name: Time Synchronization Play
hosts: servers
roles:
  - rhel-system-roles.timesync
tasks:
  - name: Set timezone
    timezone:
      name: "{{ timezone }}"
```

Cette structure sépare nettement le rôle, le code du playbook et les paramètres de configuration. Le code du playbook est simple, facile à lire et ne devrait pas nécessiter de refactorisation complexe. Le contenu du rôle est géré et pris en charge par Red Hat. Tous les paramètres sont gérés comme des variables d'inventaire.

Cette structure prend également en charge un environnement dynamique et hétérogène. Les hôtes avec de nouvelles exigences de synchronisation de l'horloge peuvent être placés dans un nouveau groupe d'hôtes. Les variables appropriées sont définies dans un fichier YAML et placées dans le sous-répertoire approprié **group\_vars** (ou **host\_vars**).

## EXEMPLE DE RÔLE SELINUX

Autre exemple, le rôle `rhel-system-roles.selinux` simplifie la gestion des paramètres de configuration de SELinux. Il est mis en œuvre à l'aide des modules Ansible associés à SELinux. L'utilisation de ce rôle présente l'avantage de vous décharger de la responsabilité d'écrire vos propres tâches. À la place, vous fournissez des variables au rôle pour le configurer, et le code mis à jour dans le rôle garantit que la configuration SELinux souhaitée est appliquée.

Parmi les tâches que ce rôle peut effectuer :

- définir le mode d'exécution ou d'autorisation ;
- exécuter **restorecon** sur des portions de la hiérarchie du système de fichiers ;
- définir des valeurs booléennes SELinux ;
- définir les contextes de fichier SELinux de manière persistante ;
- définir les mappages d'utilisateurs SELinux.

## Appel du rôle SELinux

Parfois, le rôle SELinux doit s'assurer que les hôtes gérés sont redémarrés pour appliquer complètement ses modifications. Cependant, il ne redémarre jamais lui-même les hôtes. Cela vous permet de contrôler la manière dont le redémarrage est géré. Toutefois, cela signifie que l'utilisation correcte de ce rôle dans un play s'avère un peu plus compliquée qu'à l'accoutumée.

Concrètement, le rôle définit une variable booléenne, `selinux_reboot_required`, sur **true** et échoue si un redémarrage est nécessaire. Vous pouvez utiliser une structure **block/rescue** de récupération après l'échec, en faisant échouer le play si cette variable n'est pas définie sur **true** ou en redémarrant l'hôte géré et en réexécutant le rôle si elle est sur **true**. Le bloc dans votre play devrait ressembler à ceci :

```
- 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
```

## Configuration du rôle SELinux

Les variables utilisées pour configurer le rôle `rhel-system-roles.selinux` sont documentées dans le fichier **README.md**. Les exemples suivants montrent certaines manières d'utiliser ce rôle.

La variable `selinux_state` définit le mode dans lequel SELinux est exécuté. Elle peut être définie sur **enforcing**, **permissive** ou **disabled**. Si elle n'est pas définie, le mode n'est pas modifié.

```
selinux_state: enforcing
```

La variable `selinux_booleans` prend une liste de valeurs booléennes SELinux à ajuster. Chaque élément de la liste est un hash/dictionnaire de variables : le `name` de la valeur booléenne, le `state` (qu'il soit sur **on** ou **off**), et si oui ou non le paramètre doit être **persistent** entre les redémarrages.

Cet exemple définit `httpd_enable_homedirs` sur **on** de manière persistante :

```
selinux_booleans:
  - name: 'httpd_enable_homedirs'
    state: 'on'
    persistent: 'yes'
```

La variable `selinux_fcontext` prend une liste de contextes de fichiers à définir de manière persistante (ou à supprimer). Son fonctionnement ressemble à celui de la commande **selinux fcontext**.

L'exemple suivant garantit que la stratégie dispose d'une règle pour définir le type SELinux par défaut pour tous les fichiers sous `/srv/www` sur `httpd_sys_content_t`.

```
selinux_fcontexts:
  - target: '/srv/www(.*)?'
    setype: 'httpd_sys_content_t'
    state: 'present'
```

La variable `selinux_restore_dirs` spécifie une liste de répertoires sur lesquels exécuter **restorecon**:

```
selinux_restore_dirs:
  - /srv/www
```

La variable `selinux_ports` prend une liste de ports qui devraient avoir un type SELinux spécifique.

```
selinux_ports:
  - ports: '82'
    setype: 'http_port_t'
    proto: 'tcp'
    state: 'present'
```

Il existe d'autres variables et options pour ce rôle. Reportez-vous au fichier **README.md** pour plus d'informations.



## RÉFÉRENCES

### Rôles système Red Hat Enterprise Linux (RHEL)

<https://access.redhat.com/articles/3050101>

### Rôles système Linux

<https://linux-system-roles.github.io/>

## ► EXERCICE GUIDÉ

# RÉUTILISATION DU CONTENU AVEC DES RÔLES SYSTÈME

Dans cet exercice, vous utiliserez l'un des rôles système Red Hat Enterprise Linux en association avec une tâche normale pour configurer la synchronisation de l'horloge et le fuseau horaire sur vos hôtes gérés.

## RÉSULTATS

Vous devez pouvoir :

- Installer les rôles système Red Hat Enterprise Linux.
- Rechercher et utiliser la documentation sur les rôles système RHEL.
- Utiliser le rôle `rhel-system-roles.timesync` dans un playbook pour configurer la synchronisation de l'horloge sur des hôtes distants.

## PRÉSENTATION DU SCÉNARIO

Votre organisation gère deux datacenters : l'un aux États-Unis (Chicago) et l'autre en Finlande (Helsinki). Pour faciliter l'analyse des journaux des serveurs de base de données dans les datacenters, assurez-vous que l'horloge système de chaque hôte est synchronisée à l'aide du protocole de synchronisation de l'horloge NTP (Network Time Protocol). Pour permettre l'analyse de l'activité time-of-day dans les datacenters, assurez-vous que chaque serveur de base de données a un fuseau horaire qui correspond à l'emplacement du datacenter de l'hôte.

La synchronisation de l'horloge doit respecter les exigences suivantes :

- Utilisez le serveur NTP situé à `classroom.example.com`. Activez l'option `iburst` pour accélérer la synchronisation de l'horloge initiale.
- Utilisez le paquetage `chrony` pour la synchronisation de l'horloge.

À partir de `workstation`, exécutez la commande `lab role-system setup` afin de préparer l'environnement pour cet exercice. Le script crée le répertoire de travail `role-system` et l'approvisionne à l'aide d'un fichier de configuration Ansible et d'un inventaire d'hôtes.

```
[student@workstation ~]$ lab role-system setup
```

- 1. Connectez-vous à votre hôte `workstation` en tant que `student`. Accédez au répertoire de travail `role-system`.

```
[student@workstation ~]$ cd ~/role-system
[student@workstation role-system]$
```

- 2. Installez les rôles système Red Hat Enterprise Linux sur le nœud de contrôle `workstation.lab.example.com`. Vérifiez l'emplacement d'installation des rôles sur le nœud de contrôle.
- 2.1. Utilisez la commande **ansible-galaxy** pour vérifier qu'aucun rôle n'est initialement disponible pour une utilisation dans le projet du playbook.

```
[student@workstation role-system]$ ansible-galaxy list  
[student@workstation role-system]$
```

La commande **ansible-galaxy** recherche les rôles dans trois répertoires, comme l'indique l'entrée `roles_path` du fichier **ansible.cfg**:

- **./roles**
- **/usr/share/ansible/roles**
- **/etc/ansible/roles**

La sortie ci-dessus indique qu'il n'y a aucun rôle dans aucun de ces répertoires.

- 2.2. Activez le référentiel `rhel-7-server-extras-rpms`.

```
[student@workstation role-system]$ sudo yum-config-manager \  
> --enable rhel-7-server-extras-rpms
```

- 2.3. Installez le paquetage `rhel-system-roles`.

```
[student@workstation role-system]$ sudo yum install rhel-system-roles
```

Entrez **y** lorsque vous êtes invité à installer le paquetage.

- 2.4. Utilisez la commande **ansible-galaxy** pour vérifier que les rôles système sont maintenant disponibles.

```
[student@workstation role-system]$ 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)
```

Les rôles se trouvent dans le répertoire **/usr/share/ansible/roles**. Tout rôle commençant par **linux-system-roles** est en fait un lien symbolique vers le rôle **rhel-system-roles** correspondant.

- 3. Créez un playbook, **configure\_time.yml**, avec un play ciblant le groupe d'hôtes **database\_servers**. Incluez le rôle **rhel-system-roles.timesync** rôle dans la section **roles** du play.

Le contenu de **configure\_time.yml** correspond maintenant à ce qui suit :

```
---
- name: Time Synchronization
  hosts: database_servers

  roles:
    - rhel-system-roles.timesync
```

- 4. La documentation de rôle contient une description de chaque variable de rôle, y compris la valeur par défaut de la variable. Déterminez les variables de rôle à remplacer pour répondre aux exigences de synchronisation de l'horloge.

Placez les valeurs de variable de rôle dans un fichier nommé **timesync.yml**. Comme ces valeurs de variable s'appliquent à tous les hôtes de l'inventaire, placez le fichier **timesync.yml** dans le sous-répertoire **group\_vars/all**.

- 4.1. Examinez la section *Role Variables* du fichier **README.md** pour le rôle **rhel-system-roles.timesync**.

```
[student@workstation role-system]$ cat \
> /usr/share/doc/rhel-system-roles-1.0/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
```

- 4.2. Créez le sous-répertoire **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. Créez le fichier **group\_vars/all/timesync.yml** à l'aide de l'éditeur de texte. Ajoutez des définitions de variable pour répondre aux exigences de synchronisation de l'horloge. Le fichier contient désormais les éléments suivants :

```
---
#rhel-system-roles.timesync variables for all hosts

timesync_ntp_provider: chrony

timesync_ntp_servers:
  - hostname: classroom.example.com
    iburst: yes
```

- ▶ 5. Insérez une tâche pour définir le fuseau horaire de chaque hôte. Assurez-vous que la tâche utilise le module `timezone` et s'exécute après le rôle `rhel-system-roles.timesync`.

Dans la mesure où les hôtes n'appartiennent pas au même fuseau horaire, utilisez une variable (`host_timezone`) pour le nom du fuseau horaire.

- 5.1. Examinez la section *Exemples* de la documentation du module `timezone`.

```
[student@workstation role-system]$ ansible-doc timezone | grep -A 4 "EXAMPLES"
EXAMPLES:
- name: set timezone to Asia/Tokyo
  timezone:
    name: Asia/Tokyo
```

- 5.2. Ajoutez une tâche dans la section **post\_tasks** du play dans le playbook **configure\_time.yml**. Modélez la tâche d'après l'exemple de la documentation, mais utilisez la variable `host_timezone` pour le nom du fuseau horaire.

La documentation du module `timezone` recommande également de redémarrer le service `crond` après avoir modifié le fuseau horaire. Ajoutez un mot-clé **notify** à la tâche, avec une valeur associée **restart crond**. La section **post\_tasks** correspond à ce qui suit :

```
post_tasks:
  - name: Set timezone
    timezone:
      name: "{{ host_timezone }}"
    notify: restart crond
```

- 5.3. Ajoutez le gestionnaire **restart crond** au play **Time Synchronization**. Le playbook complet contient désormais les éléments suivants :

```
---
- name: Time Synchronization
  hosts: database_servers

  roles:
    - rhel-system-roles.timesync
```

```

post_tasks:
  - name: Set timezone
    timezone:
      name: "{{ host_timezone }}"
    notify: restart crond

handlers:
  - name: restart crond
    service:
      name: crond
      state: restarted

```

- 6. Pour chaque datacenter, créez un fichier nommé **timezone.yml** contenant une valeur appropriée pour la variable `host_timezone`. Utilisez la commande **timedatectl list-timezones** pour trouver la chaîne de fuseau horaire valide pour chaque datacenter.
- 6.1. Créez les sous-répertoires **group\_vars** pour les groupes d'hôtes na-datacenter et 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. Utilisez la commande **timedatectl list-timezones** pour déterminer le fuseau horaire des datacenters américains et européens :

```
[student@workstation role-system]$ timedatectl list-timezones | grep Chicago
America/Chicago
[student@workstation role-system]$ timedatectl list-timezones | grep Helsinki
Europe/Helsinki
```

- 6.3. Créez le fichier **timezone.yml** pour les deux datacenters :

```
[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. Exécutez le playbook.

```
[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]

RUNNING HANDLER [rhel-system-roles.timesync : restart chronyrd] ****
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 [restart crond] ****
changed: [serverb.lab.example.com]
changed: [servera.lab.example.com]

PLAY RECAP ****
servera.lab.example.com      : ok=17    changed=6    unreachable=0    failed=0
serverb.lab.example.com      : ok=17    changed=6    unreachable=0    failed=0

```

- 8. Vérifiez les paramètres de fuseau horaire de chaque serveur. Utilisez une commande ad hoc Ansible pour afficher la sortie de la commande **date** sur tous les serveurs de base de données.

```

[student@workstation role-system]$ ansible database_servers -m shell -a date
serverb.lab.example.com | CHANGED | rc=0 >>
Tue Nov 27 23:24:27 EET 2018

servera.lab.example.com | CHANGED | rc=0 >>
Tue Nov 27 15:24:27 CST 2018

```

Chaque serveur est doté d'un paramètre de fuseau horaire basé sur son emplacement géographique.

## Nettoyage

Exécutez la commande **lab role-system cleanup** pour nettoyer l'hôte géré.

```
[student@workstation ~]$ lab role-system cleanup
```

L'exercice guidé est maintenant terminé.

## ► OPEN LAB

# MISE EN ŒUVRE DES RÔLES

## LISTE DE CONTRÔLE DES PERFORMANCES

Au cours de cet atelier, vous allez créer des rôles Ansible qui utilisent des variables, des fichiers, des modèles, des tâches et des gestionnaires.

## RÉSULTATS

Vous devez pouvoir :

- Créer des rôles Ansible utilisant des variables, des fichiers, des modèles, des tâches et des gestionnaires pour configurer un serveur Web de développement.
- Utiliser un rôle hébergé dans un référentiel distant dans un playbook.
- Utiliser un rôle système Red Hat Enterprise Linux dans un playbook.

## PRÉSENTATION DU SCÉNARIO

Votre entreprise doit fournir un serveur Web unique pour héberger le code de développement pour tous les développeurs Web. Vous êtes chargé d'écrire un playbook pour configurer ce serveur Web de développement.

Le serveur Web de développement doit répondre à plusieurs exigences :

- La configuration du serveur de développement correspond à celle du serveur de production. Le serveur de production est configuré à l'aide d'un rôle Ansible, développé par l'équipe d'infrastructure de l'organisation.
- Chaque développeur se voit attribuer un répertoire sur le serveur de développement pour héberger le code et le contenu. Le contenu de chaque développeur est accessible via un port non standard attribué.
- SELinux est configuré pour exécuter et cibler.

Votre playbook effectuera les éléments suivants :

- Utiliser un rôle pour configurer les répertoires et les ports pour chaque développeur sur le serveur Web. Vous devez écrire ce rôle.

Ce rôle a une dépendance sur un rôle écrit par l'organisation pour configurer Apache. Vous devez définir la dépendance à l'aide de la version **v1.4** du rôle organisationnel. L'URL du référentiel de dépendance est :`git@workstation.lab.example.com:infra/apache`

- Utiliser le rôle `rhel-system-roles.selinux` pour configurer SELinux pour les ports HTTP non standard utilisés par votre serveur Web. Vous recevrez un fichier de variable `selinux.yml` qui peut être installé en tant que fichier `group_vars` pour transmettre les paramètres corrects au rôle.

Connectez-vous en tant qu'utilisateur **student** sur `workstation`, puis exécutez `lab role-review setup`. Le script crée le répertoire de projet nommé `role-review` et l'approvisionne

à l'aide du fichier de configuration Ansible, de l'inventaire d'hôtes et d'autres fichiers d'exercice pratique.

```
[student@workstation ~]$ lab role-review setup
```

1. Connectez-vous à votre hôte `workstation` en tant que `student`. Accédez au répertoire de travail `role-review`.

2. Le groupe d'hôtes du serveur Web de développement doit être `dev_webserver`. Confirmez que ce groupe d'hôtes existe dans le fichier `inventory`.

3. Créez un playbook nommé `web_dev_server.yml` avec un seul play nommé **Configure Dev Web Server**. Configurez le play pour qu'il cible le groupe d'hôtes `dev_webserver`. N'ajoutez aucun rôle ni aucune tâche au play pour le moment.

Assurez-vous que le play force l'exécution des gestionnaires, car vous pourriez rencontrer une erreur lors du développement du playbook.

4. Vérifiez la syntaxe du playbook. Exécutez le playbook. La vérification de la syntaxe doit être validée et le playbook doit s'exécuter correctement.

5. Assurez-vous que les dépendances de rôle du play sont installées.

Le rôle `apache.developer_configs` que vous allez créer dépend du rôle `infra.apache`. Créez un fichier `roles/requirements.yml`. Il doit installer le rôle à partir du référentiel Git à l'emplacement `git@workstation.lab.example.com:infra/apache`, utiliser la version **v1.4** et le nommer `infra.apache` en local. Vous pouvez supposer que vos clés SSH sont configurées pour vous permettre d'obtenir automatiquement les rôles à partir de ce référentiel. Installez le rôle avec la commande `ansible-galaxy`.

De plus, installez le paquetage `rhel-system-roles`.

6. Initialisez un nouveau rôle nommé `apache.developer_configs` dans le sous-répertoire `roles`.

Ajoutez le rôle `infra.apache` en tant que dépendance du nouveau rôle, en utilisant les mêmes informations pour le nom, la source, la version et le système de contrôle de version que le fichier `roles/requirements.yml`.

Le fichier `developer_tasks.yml` du répertoire du projet contient des tâches pour le rôle. Déplacez ce fichier à l'emplacement correct afin qu'il soit le fichier de tâches pour ce rôle.

Le fichier `developer.conf.j2` dans le répertoire du projet est un modèle Jinja2 utilisé par le fichier de tâches. Déplacez-le à l'emplacement approprié pour les fichiers de modèle utilisés par ce rôle.

7. Le rôle `apache.developer_configs` traitera une liste d'utilisateurs définie dans une variable nommée `web_developers`. Le fichier `web_developers.yml` du répertoire du projet définit la variable de liste d'utilisateurs `web_developers`. Examinez ce fichier et utilisez-le pour définir la variable `web_developers` pour le groupe d'hôtes du serveur Web de développement.

8. Ajoutez le rôle `apache.developer_configs` au play dans le playbook `web_dev_server.yml`.

9. Vérifiez la syntaxe du playbook. Exécutez le playbook. La vérification de la syntaxe doit être validée, mais le playbook doit échouer lorsque le rôle `infra.apache` tente de redémarrer Apache HTTPD.

10. Apache HTTPD n'a pas pu redémarrer à l'étape précédente, car les ports réseau utilisés par vos développeurs sont identifiés avec les contextes SELinux incorrects. Le fichier de variable `selinux.yml` vous a été fourni ; il peut être utilisé avec le rôle `rhel-system-roles.selinux` pour résoudre le problème.

Créez une section **pre\_tasks** pour votre play dans le playbook **web\_dev\_server.yml**. Dans cette section, utilisez une tâche pour inclure le rôle **rhel-system-roles.selinux** dans une structure **block/rescue** de sorte qu'il soit correctement appliqué. Consultez le cours ou la documentation sur ce rôle pour voir comment procéder.

Examinez le fichier **selinux.yml**. Déplacez-le à l'emplacement correct afin que ses variables soient définies pour le groupe d'hôtes **dev\_webserver**.

11. Vérifiez la syntaxe du playbook final. La vérification de la syntaxe doit être validée.
12. Exécutez le playbook. Il doit être correctement exécuté.
13. Testez la configuration du serveur Web de développement. Vérifiez que tous les points d'accès sont accessibles et qu'ils servent le contenu de chaque développeur.

## Évaluation

Notez votre travail en exécutant la commande **lab role-review grade** à partir de votre poste de travail **workstation**. Corrigez toute erreur signalée et répétez le script tant que des erreurs persistent.

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

## Nettoyage

À partir de **workstation**, exécutez le script **lab role-review cleanup** pour effacer l'exercice.

```
[student@workstation ~]$ lab role-review cleanup
```

L'atelier est maintenant terminé.

## ► SOLUTION

# MISE EN ŒUVRE DES RÔLES

## LISTE DE CONTRÔLE DES PERFORMANCES

Au cours de cet atelier, vous allez créer des rôles Ansible qui utilisent des variables, des fichiers, des modèles, des tâches et des gestionnaires.

## RÉSULTATS

Vous devez pouvoir :

- Créer des rôles Ansible utilisant des variables, des fichiers, des modèles, des tâches et des gestionnaires pour configurer un serveur Web de développement.
- Utiliser un rôle hébergé dans un référentiel distant dans un playbook.
- Utiliser un rôle système Red Hat Enterprise Linux dans un playbook.

## PRÉSENTATION DU SCÉNARIO

Votre entreprise doit fournir un serveur Web unique pour héberger le code de développement pour tous les développeurs Web. Vous êtes chargé d'écrire un playbook pour configurer ce serveur Web de développement.

Le serveur Web de développement doit répondre à plusieurs exigences :

- La configuration du serveur de développement correspond à celle du serveur de production. Le serveur de production est configuré à l'aide d'un rôle Ansible, développé par l'équipe d'infrastructure de l'organisation.
- Chaque développeur se voit attribuer un répertoire sur le serveur de développement pour héberger le code et le contenu. Le contenu de chaque développeur est accessible via un port non standard attribué.
- SELinux est configuré pour exécuter et cibler.

Votre playbook effectuera les éléments suivants :

- Utiliser un rôle pour configurer les répertoires et les ports pour chaque développeur sur le serveur Web. Vous devez écrire ce rôle.

Ce rôle a une dépendance sur un rôle écrit par l'organisation pour configurer Apache. Vous devez définir la dépendance à l'aide de la version **v1.4** du rôle organisationnel. L'URL du référentiel de dépendance est : `git@workstation.lab.example.com:infra/apache`

- Utiliser le rôle `rhel-system-roles.selinux` pour configurer SELinux pour les ports HTTP non standard utilisés par votre serveur Web. Vous recevrez un fichier de variable `selinux.yml` qui peut être installé en tant que fichier `group_vars` pour transmettre les paramètres corrects au rôle.

Connectez-vous en tant qu'utilisateur **student** sur `workstation`, puis exécutez `lab role-review setup`. Le script crée le répertoire de projet nommé `role-review` et l'approvisionne

**CHAPITRE 8** | Simplification des playbooks avec des rôles

à l'aide du fichier de configuration Ansible, de l'inventaire d'hôtes et d'autres fichiers d'exercice pratique.

```
[student@workstation ~]$ lab role-review setup
```

1. Connectez-vous à votre hôte `workstation` en tant que `student`. Accédez au répertoire de travail `role-review`.

```
[student@workstation ~]$ cd ~/role-review  
[student@workstation role-review]$
```

2. Le groupe d'hôtes du serveur Web de développement doit être `dev_webserver`. Confirmez que ce groupe d'hôtes existe dans le fichier `inventory`.

```
[student@workstation role-review]$ cat inventory  
[controlnode]  
workstation.lab.example.com  
  
[dev_webserver]  
servera.lab.example.com
```

3. Créez un playbook nommé `web_dev_server.yml` avec un seul play nommé **Configure Dev Web Server**. Configurez le play pour qu'il cible le groupe d'hôtes `dev_webserver`. N'ajoutez aucun rôle ni aucune tâche au play pour le moment.

Assurez-vous que le play force l'exécution des gestionnaires, car vous pourriez rencontrer une erreur lors du développement du playbook.

Une fois complet, le playbook `/home/student/role-review/web_dev_server.yml` contient les éléments suivants :

```
---  
- name: Configure Dev Web Server  
  hosts: dev_webserver  
  force_handlers: yes
```

4. Vérifiez la syntaxe du playbook. Exécutez le playbook. La vérification de la syntaxe doit être validée et le playbook doit s'exécuter correctement.

```
[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
```

5. Assurez-vous que les dépendances de rôle du play sont installées.

Le rôle `apache.developer_configs` que vous allez créer dépend du rôle `infra.apache`. Créez un fichier `roles/requirements.yml`. Il doit installer le rôle à partir du référentiel Git à l'emplacement `git@workstation.lab.example.com:infra/apache`, utiliser la version `v1.4` et le nommer `infra.apache` en local. Vous pouvez supposer que vos clés SSH sont configurées pour vous permettre d'obtenir automatiquement les rôles à partir de ce référentiel. Installez le rôle avec la commande `ansible-galaxy`.

De plus, installez le paquetage `rhel-system-roles`.

- Créez un sous-répertoire `roles` pour le projet du playbook.

```
[student@workstation role-review]$ mkdir -v roles
mkdir: created directory 'roles'
```

- Créez un fichier `roles/requirements.yml` et ajoutez une entrée pour le rôle `infra.apache`. Utilisez la version `v1.4` du référentiel Git du rôle.

Une fois complet, le fichier `roles/requirements.yml` contient les éléments suivants :

```
- name: infra.apache
  src: git@workstation.lab.example.com:infra/apache
  scm: git
  version: v1.4
```

- Installez les dépendances du projet.

```
[student@workstation role-review]$ ansible-galaxy install \
> -r requirements.yml -p roles
- extracting infra.apache to /home/student/role-review/roles/infra.apache
- infra.apache (v1.4) was installed successfully
```

- Installez le paquetage des rôles système RHEL.

```
[student@workstation role-review]$ sudo yum install rhel-system-roles
```

- Initialisez un nouveau rôle nommé `apache.developer_configs` dans le sous-répertoire `roles`.

Ajoutez le rôle `infra.apache` en tant que dépendance du nouveau rôle, en utilisant les mêmes informations pour le nom, la source, la version et le système de contrôle de version que le fichier `roles/requirements.yml`.

Le fichier `developer_tasks.yml` du répertoire du projet contient des tâches pour le rôle. Déplacez ce fichier à l'emplacement correct afin qu'il soit le fichier de tâches pour ce rôle.

Le fichier `developer.conf.j2` dans le répertoire du projet est un modèle Jinja2 utilisé par le fichier de tâches. Déplacez-le à l'emplacement approprié pour les fichiers de modèle utilisés par ce rôle.

- Utilisez `ansible-galaxy init` pour créer un squelette de rôle pour le rôle `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]$
```

- 6.2. Mettez à jour le fichier **roles/apache.developer\_configs/meta/main.yml** du rôle apache.developer\_configs afin de refléter une dépendance sur le rôle **infra.apache**.

Après édition, la variable **dependencies** est définie comme suit :

```
dependencies:  
  - name: infra.apache  
    src: git@workstation.lab.example.com:infra/apache  
    scm: git  
    version: v1.4
```

Utilisez **grep** pour confirmer le contenu correct de la variable **dependencies**:

```
[student@workstation role-review]$ cd roles/apache.developer_configs  
[student@workstation apache.developer_configs]$ grep -A4 \  
> dependencies: meta/main.yml  
dependencies:  
  - name: infra.apache  
    src: git@workstation.lab.example.com:infra/apache  
    scm: git  
    version: v1.4  
[student@workstation apache.developer_configs]$ cd /home/student/role-review  
[student@workstation role-review]$
```

- 6.3. Remplacez le fichier **tasks/main.yml** du rôle par le fichier **developer\_tasks.yml**.

```
[student@workstation role-review]$ mv -v developer_tasks.yml \  
> roles/apache.developer_configs/tasks/main.yml  
'developer_tasks.yml' -> 'roles/apache.developer_configs/tasks/main.yml'
```

- 6.4. Placez le fichier **developer.conf.j2** dans le répertoire **templates** du rôle.

```
[student@workstation role-review]$ mv -v developer.conf.j2 \  
> roles/apache.developer_configs/templates/  
'developer.conf.j2' -> 'roles/apache.developer_configs/templates/  
developer.conf.j2'
```

7. Le rôle **apache.developer\_configs** traitera une liste d'utilisateurs définie dans une variable nommée **web\_developers**. Le fichier **web\_developers.yml** du répertoire du projet définit la variable de liste d'utilisateurs **web\_developers**. Examinez ce fichier et utilisez-le pour définir la variable **web\_developers** pour le groupe d'hôtes du serveur Web de développement.

- 7.1. Examinez le fichier **web\_developers.yml**.

---

```
web_developers:  
  - username: jdoe  
    name: John Doe  
    user_port: 9081  
  - username: jdoe2  
    name: Jane Doe  
    user_port: 9082
```

name, username, user\_port sont définis pour chaque développeur Web.

- 7.2. Placez le fichier **web\_developers.yml** dans le sous-répertoire **group\_vars/dev\_webserver**.

```
[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  
'developers.yml' -> 'group_vars/dev_webserver/developers.yml'
```

8. Ajoutez le rôle apache.developer\_configs au play dans le playbook **web\_dev\_server.yml**.

Voici le playbook édité :

```
---  
- name: Configure Dev Web Server  
  hosts: dev_webserver  
  force_handlers: yes  
  roles:  
    - apache.developer_configs
```

9. Vérifiez la syntaxe du playbook. Exécutez le playbook. La vérification de la syntaxe doit être validée, mais le playbook doit échouer lorsque le rôle **infra.apache** tente de redémarrer Apache HTTPD.

```
[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={'username': u'jdoe', 'user_port': 9081, 'name': u'John Doe'})
changed: [servera.lab.example.com] => (item={'username': u'jdoe2', 'user_port': 9082, '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=7    unreachable=0    failed=1

```

Une erreur se produit lorsque le service `httpd` est redémarré. Le démon de service `httpd` ne peut pas se lier aux ports HTTP non standard, en raison du contexte SELinux sur ces ports.

10. Apache HTTPD n'a pas pu redémarrer à l'étape précédente, car les ports réseau utilisés par vos développeurs sont identifiés avec les contextes SELinux incorrects. Le fichier de variable **`selinux.yml`** vous a été fourni ; il peut être utilisé avec le rôle `rhel-system-roles.selinux` pour résoudre le problème.

Créez une section **`pre_tasks`** pour votre play dans le playbook **`web_dev_server.yml`**. Dans cette section, utilisez une tâche pour inclure le rôle `rhel-system-roles.selinux` dans une structure `block/rescue` de sorte qu'il soit correctement appliqué. Consultez le cours ou la documentation sur ce rôle pour voir comment procéder.

Examinez le fichier **`selinux.yml`**. Déplacez-le à l'emplacement correct afin que ses variables soient définies pour le groupe d'hôtes `dev_webserver`.

- 10.1. La section **`pre_tasks`** peut être ajoutée à la fin du play dans le playbook **`web_dev_server.yml`**.

Vous pouvez consulter le bloc dans **`/usr/share/doc/rhel-system-roles-1.0/selinux/example-selinux-playbook.yml`** pour obtenir un aperçu de la façon d'appliquer le rôle, mais Red Hat Ansible Engine 2.7 vous permet de remplacer le `shell` complexe et la logique `wait_for` par le module `reboot`.

La section **`pre_tasks`** doit contenir les éléments suivants :

```

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

```

- 10.2. Le fichier **selinux.yml** contient les définitions de variable du rôle **rhel-system-roles.selinux**. Utilisez le fichier pour définir les variables du groupe d'hôtes du play.

```

[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/
'selinux.yml' -> 'group_vars/dev_webserver/selinux.yml'

```

11. Vérifiez la syntaxe du playbook final. La vérification de la syntaxe doit être validée.

```

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

playbook: web_dev_server.yml
[student@workstation role-review]$

```

Le playbook **web\_dev\_server.yml** doit se présenter comme suit :

```

---
- 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
```



### NOTE

Que la section **pre\_tasks** se situe à la fin du play ou dans une position « correcte » en termes d'ordre d'exécution dans le fichier de playbook n'a aucune importance pour **ansible-playbook**. Il exécutera toujours les tâches du play dans le bon ordre.

12. Exéutez le playbook. Il doit être correctement exécuté.

```
[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 python2 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={u'username': u'jdoe', u'user_port': 9081,
  u'name': u'John Doe'})
ok: [servera.lab.example.com] => (item={u'username': u'jdoe2', u'user_port': 9082,
  u'name': u'Jane Doe'})

PLAY RECAP ****
servera.lab.example.com      : ok=18    changed=3    unreachable=0    failed=0
```

13. Testez la configuration du serveur Web de développement. Vérifiez que tous les points d'accès sont accessibles et qu'ils servent le contenu de chaque développeur.

```
[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]$
```

## Évaluation

Notez votre travail en exécutant la commande **lab role-review grade** à partir de votre poste de travail `workstation`. Corrigez toute erreur signalée et répétez le script tant que des erreurs persistent.

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

## Nettoyage

À partir de `workstation`, exécutez le script **lab role-review cleanup** pour effacer l'exercice.

```
[student@workstation ~]$ lab role-review cleanup
```

L'atelier est maintenant terminé.

# RÉSUMÉ

---

Dans ce chapitre, vous avez appris les principes suivants :

- Les rôles organisent le code Ansible de sorte qu'il puisse être réutilisé et partagé.
- Les variables de rôle définies dans **defaults/main.yml** sont remplacées par des variables d'inventaire ou de play. Celles qui figurent dans **vars/main.yml** sont utilisées en interne par le rôle.
- Si vous fournissez une liste de rôles dans la section **roles** d'un play, ils sont exécutés avant les tâches de la section **tasks** de votre play. Tous les gestionnaires qu'ils notifient sont exécutés après les tâches dans l'exécution de **tasks**.
- Les tâches de la section **pre\_tasks** sont exécutées et exécutent les gestionnaires notifiés avant les **roles** et les **tasks**. Les tâches de la section **post\_tasks** sont exécutées et exécutent les gestionnaires notifiés après les **roles** et les **tasks**.
- Les versions récentes d'Ansible vous permettent d'inclure ou d'importer un rôle en tant que tâche avec les modules `include_role` et `import_role`.
- Ansible Galaxy [<https://galaxy.ansible.com>] est une bibliothèque publique de rôles Ansible créés par des utilisateurs Ansible.
- La commande **ansible-galaxy** permet de rechercher, d'installer, de lister, de supprimer ou d'initialiser des rôles ou d'afficher des informations sur les rôles.
- Les rôles externes nécessaires à un playbook peuvent être définis dans le fichier **roles/requirements.yml**. La commande **ansible-galaxy install -r roles/requirements.yml** utilise ce fichier pour installer les rôles sur le nœud de contrôle.
- Red Hat Enterprise Linux System Roles est un ensemble de rôles testés et pris en charge destinés à vous aider à configurer les sous-systèmes hôte dans les différentes versions de Red Hat Enterprise Linux.
- Le paramètre par défaut `roles_path` trouve automatiquement les rôles système RHEL installés dans le répertoire **/usr/share/ansible/roles**.



## CHAPITRE 9

# RÉSOLUTION DES PROBLÈMES LIÉS À ANSIBLE

### PROJET

Résoudre les problèmes liés aux playbooks et hôtes gérés.

### OBJECTIFS

- Résoudre les problèmes génériques avec un nouveau playbook et les réparer.
- Résoudre les échecs sur les hôtes gérés lors de l'exécution d'un playbook.

### SECTIONS

- Résolution des problèmes liés aux playbooks (et exercice guidé)
- Résolution des problèmes liés aux hôtes gérés Ansible (et exercice guidé)

### ATELIER

- Résolution des problèmes liés à Ansible

# RÉSOLUTION DE PROBLÈMES CONCERNANT LES PLAYBOOKS

---

## OBJECTIFS

Après avoir terminé cette section, les stagiaires doivent pouvoir résoudre les problèmes génériques d'un nouveau playbook.

**Figure 9.0: Résolution de problèmes concernant les playbooks Ansible**

## FICHIERS JOURNAUX D'ANSIBLE

Par défaut, Red Hat Ansible Engine n'est pas configuré pour consigner ses résultats dans un fichier journal, mais il contient une infrastructure de journalisation intégrée qui peut être configurée à l'aide du paramètre `log_path` dans la section `default` du fichier de configuration `ansible.cfg` ou à l'aide de la variable d'environnement `$ANSIBLE_LOG_PATH`. Si le paramètre et/ou la variable sont configurés, Ansible stocke les résultats des commandes `ansible` et `ansible-playbook` dans le fichier journal paramétré via le fichier de configuration `ansible.cfg` ou la variable d'environnement `$ANSIBLE_LOG_PATH`.

Si les fichiers journaux Ansible doivent être conservés dans le répertoire des fichiers journaux par défaut `/var/log`, alors les playbooks doivent être exécutés dans une session `root` ou les permissions sur `/var/log` doivent être mises à jour. Le plus souvent, les fichiers journaux sont créés dans le répertoire de playbook local.



### NOTE

Si vous configurez Ansible afin qu'il écrive des fichiers journaux dans `/var/log`, Red Hat vous recommande de configurer `logrotate` pour gérer les fichiers journaux Ansible.

## LE MODULE DEBUG

L'un des modules disponibles dans Ansible, le module `debug`, peut offrir un meilleur aperçu des événements qui se produisent dans le play. Ce module peut afficher la valeur d'une variable donnée à un moment précis du play. Cette fonction est déterminante dans le débogage des tâches qui utilisent des variables pour communiquer ensemble (par exemple, en utilisant la sortie d'une tâche comme entrée de la tâche suivante).

Les exemples suivants utilisent les paramètres `msg` et `var` à l'intérieur des tâches `debug`. Le premier exemple affiche la valeur au moment de l'exécution du fait `ansible_facts['memfree_mb']` dans le cadre d'un message imprimé à la sortie de `ansible-playbook`. Le deuxième exemple affiche la valeur de la variable `output`.

```
- name: Display free memory
  debug:
    msg: "The free memory for this system is {{ ansible_facts['memfree_mb'] }}"
```

```
- name: Display the "output" variable
  debug:
```

```
var: output
verbosity: 2
```

## GESTION DES ERREURS

Plusieurs problèmes peuvent survenir lors de l'exécution d'un playbook, notamment au niveau de la syntaxe du playbook lui-même ou des modèles qu'il utilise. Mais des problèmes de connectivité peuvent également se produire avec les hôtes gérés (comme une erreur dans le nom d'un hôte géré dans le fichier d'inventaire). Ces erreurs sont générées par la commande **ansible-playbook** au moment de l'exécution.

L'option **--syntax-check** qui vérifie la syntaxe YAML du playbook vous a été présentée précédemment dans ce cours. Il est recommandé d'exécuter une vérification de la syntaxe sur votre playbook avant de l'utiliser ou si vous rencontrez des problèmes.

```
[student@demo ~]$ ansible-playbook play.yml --syntax-check
```

Vous pouvez également utiliser l'option **--step** pour parcourir un playbook, une tâche après l'autre. La commande **ansible-playbook --step** invite de manière interactive à confirmer que vous souhaitez que chaque tâche soit exécutée.

```
[student@demo ~]$ ansible-playbook play.yml --step
```

L'option **--start-at-task** vous permet de lancer l'exécution d'un playbook à partir d'une tâche spécifique. Elle prend comme argument le nom de la tâche à partir de laquelle démarrer.

```
[student@demo ~]$ ansible-playbook play.yml --start-at-task="start httpd service"
```

## DÉBOGAGE AVEC ANSIBLE-PLAYBOOK

Le résultat généré à l'exécution d'un playbook à l'aide de la commande **ansible-playbook** est un bon point de départ pour commencer à résoudre les problèmes au niveau des hôtes gérés par Ansible. Considérez la sortie suivante d'une exécution de playbook :

```
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
```

Le résultat précédent affiche un en-tête **PLAY** accompagné du nom de play à exécuter, suivi d'un ou plusieurs en-têtes **TASK**. Chaque en-tête représente la *tâche* correspondante dans le playbook, et est exécuté sur tous les hôtes gérés appartenant au groupe inclus dans le playbook dans le paramètre *hosts*.

À mesure que chaque hôte géré exécute les tâches de chaque play, le nom de l'hôte géré apparaît sous l'en-tête **TASK** correspondant, ainsi que l'état de la tâche sur l'hôte géré en question. Les états de tâche peuvent apparaître comme **ok**, **fatal**, **changed** ou **skipping**.

Au bas de la sortie de chaque play, la section **PLAY RECAP** affiche le nombre de tâches exécutées pour chaque hôte géré.

Comme indiqué précédemment dans le cours, vous pouvez augmenter le niveau de détail de la sortie de **ansible-playbook** en ajoutant une ou plusieurs options **-v**. La commande **ansible-playbook -v** fournit des informations de débogage supplémentaires, sur quatre niveaux maximum.

#### Configuration du niveau de détail

| OPTION       | DESCRIPTION                                                                                                                                        |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>-v</b>    | Les résultats sont affichés.                                                                                                                       |
| <b>-vv</b>   | Les résultats et les données entrées sont affichés.                                                                                                |
| <b>-vvv</b>  | Inclut des informations sur les connexions aux hôtes gérés.                                                                                        |
| <b>-vvvv</b> | Inclut des informations supplémentaires, telles que les scripts exécutés sur chaque hôte distant, ainsi que l'utilisateur exécutant chaque script. |

## PRATIQUES RECOMMANDÉES POUR LA GESTION DES PLAYBOOKS

Si les outils cités précédemment permettent d'identifier et de résoudre les problèmes rencontrés dans les playbooks, lors du développement des playbooks il est important de garder à l'esprit certaines pratiques recommandées qui facilitent la résolution des problèmes. Voici certaines pratiques utiles pour développer des playbooks :

- Utilisez une description concise de l'objet du play ou de la tâche pour nommer les plays et les tâches. Le nom du play ou de la tâche est affiché au moment de l'exécution du playbook, ce qui permet également de décrire ce que chaque play ou tâche est censé accomplir, et éventuellement pourquoi il est nécessaire.
- Ajoutez des commentaires afin de compléter la documentation en ligne sur les tâches.
- Faites bon usage des espaces verticaux. En général, organisez les attributs de tâche verticalement pour les rendre plus lisibles.
- Une mise en retrait horizontale cohérente est essentielle. Utilisez des espaces, pas des tabulations, pour éviter les erreurs de mise en retrait. Configurez l'éditeur de texte pour insérer des espaces lorsque vous appuyez sur la touche de **tabulation**.
- Simplifiez au maximum le playbook. Utilisez uniquement les fonctions dont vous avez besoin.



### RÉFÉRENCES

#### Configuration d'Ansible – Documentation Ansible

[https://docs.ansible.com/ansible/2.7/installation\\_guide/intro\\_configuration.html](https://docs.ansible.com/ansible/2.7/installation_guide/intro_configuration.html)

#### debug : imprimer des instructions lors de l'exécution – Documentation Ansible

[https://docs.ansible.com/ansible/2.7/modules/debug\\_module.html](https://docs.ansible.com/ansible/2.7/modules/debug_module.html)

#### Bonnes pratiques – Documentation Ansible

[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks\\_best\\_practices.html](https://docs.ansible.com/ansible/2.7/user_guide/playbooks_best_practices.html)

## ► EXERCICE GUIDÉ

# RÉSOLUTION DE PROBLÈMES CONCERNANT LES PLAYBOOKS

Dans cet exercice, vous allez résoudre les problèmes d'un playbook qui vous a été remis et qui ne fonctionne pas correctement.

## RÉSULTATS

Vous devez pouvoir :

- Résoudre des problèmes liés aux playbooks.

Connectez-vous à **workstation** en tant qu'utilisateur **student** avec le mot de passe **student**.

Sur **workstation**, exécutez le script **lab troubleshoot-playbook setup**. Il vérifie si Ansible est installé sur **workstation**. Il crée le répertoire **/home/student/troubleshoot-playbook/** et y télécharge les fichiers **inventory**, **samba.yml** et **samba.conf.j2** à partir de <http://materials.example.com/labs/troubleshoot-playbook/>.

```
[student@workstation ~]$ lab troubleshoot-playbook setup
```

- 1. Sur **workstation**, accédez au répertoire **/home/student/troubleshoot-playbook/**.

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

- 2. Créez un fichier nommé **ansible.cfg** dans le répertoire actif. Il convient de définir le paramètre **log\_path** pour écrire des journaux Ansible dans le fichier **/home/student/troubleshoot-playbook/ansible.log**. Il convient de définir le paramètre **inventory** à utiliser pour utiliser le fichier **/home/student/troubleshoot-playbook/inventory** déployé par le script d'atelier.

Lorsque vous avez terminé, **ansible.cfg** doit contenir les éléments suivants :

```
[defaults]
log_path = /home/student/troubleshoot-playbook/ansible.log
inventory = /home/student/troubleshoot-playbook/inventory
```

- 3. Exécutez le playbook. Cela échoue et produit une erreur.

Si tout est correct, ce playbook configure un serveur Samba. Toutefois, l'exécution échoue en raison de l'absence de guillemets dans la définition de la variable **random\_var**. Lisez le message d'erreur pour voir comment **ansible-playbook** signale le problème. Notez que la variable **random\_var** comprend une valeur contenant deux points et n'est pas entre guillemets.

```
[student@workstation troubleshoot-playbook]$ ansible-playbook samba.yml
```

```
ERROR! Syntax Error while loading YAML.  
mapping values are not allowed in this context  
  
The error appears to have been 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. Vérifiez que l'erreur a été correctement consignée dans le fichier **/home/student/troubleshoot-playbook/ansible.log**.

```
[student@workstation troubleshoot-playbook]$ tail ansible.log  
The error appears to have been 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. Modifiez le playbook et corrigez l'erreur, puis ajoutez des guillemets à l'ensemble de la valeur attribuée à la variable **random\_var**. La version corrigée du playbook **samba.yml** doit contenir les éléments suivants :

```
...output omitted...  
vars:  
  install_state: installed  
  random_var: "This is colon: test"  
...output omitted...
```

- ▶ 6. Vérifiez le playbook en utilisant l'option **--syntax-check**. Une erreur est générée en raison de l'espace supplémentaire au niveau du retrait de la dernière tâche, **deliver samba config**.

```
[student@workstation troubleshoot-playbook]$ ansible-playbook --syntax-check \  
> samba.yml  
ERROR! Syntax Error while loading YAML.  
      did not find expected key  
  
The error appears to have been in '/home/student/troubleshoot-playbook/samba.yml':  
line 43, 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. Corrigez le playbook en supprimant l'espace supplémentaire sur toutes les lignes de cette tâche. Le playbook corrigé doit apparaître comme suit :

```
...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. Exécutez le playbook en utilisant l'option **--syntax-check**. Une erreur est générée, car la variable `install_state` est utilisée comme paramètre dans la tâche **install samba**. Elle n'est pas entre guillemets.

```
[student@workstation troubleshoot-playbook]$ ansible-playbook --syntax-check \
> samba.yml
ERROR! Syntax Error while loading YAML.
  found unacceptable key (unhashable type: 'AnsibleMapping')

The error appears to have been 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. Modifiez le playbook afin de corriger la tâche **install samba**. La référence à la variable `install_state` doit être placée entre guillemets. Le fichier final doit contenir les éléments suivants :

```
...output omitted...
tasks:
- name: install samba
  yum:
    name: samba
    state: "{{ install_state }}"
...output omitted...
```

- ▶ 10. Exécutez le playbook en utilisant l'option **--syntax-check**. Il ne devrait rester aucune erreur de syntaxe.

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

playbook: samba.yml
```

- ▶ 11. Exécutez le playbook. Une erreur liée à SSH est générée.

```
[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: Could not resolve hostname
servera.lab.exammples.com: Name or service not known\\r\\n", "unreachable": true}
      to retry, use: --limit @/home/student/troubleshoot-playbook/samba.retry

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

- ▶ 12. Assurez-vous que l'hôte géré `servera.lab.example.com` est exécuté, en utilisant la commande **ping**.

```
[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. Assurez-vous que vous pouvez vous connecter via SSH à l'hôte géré `servera.lab.example.com` en tant qu'utilisateur `devops` et que les clés SSH appropriées sont en place. Déconnectez-vous de nouveau lorsque vous avez terminé.

```
[student@workstation troubleshoot-playbook]$ ssh devops@servera.lab.example.com
Warning: Permanently added 'servera.lab.example.com,172.25.250.10' (ECDSA) to the
list of known hosts.
...output omitted...
[devops@servera ~]$ exit
Connection to servera.lab.example.com closed.
```

- 14. Exécutez de nouveau le playbook avec `-vvvv` afin d'obtenir de plus amples informations sur l'exécution. Une erreur est générée car l'hôte géré `servera.lab.example.com` est injoignable.

```
[student@workstation troubleshoot-playbook]$ ansible-playbook -vvvv samba.yml
ansible-playbook 2.7.1
  config file = /home/student/troubleshoot-playbook/ansible.cfg
  configured module search path = [u'/home/student/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']
    ansible python module location = /usr/lib/python2.7/site-packages/ansible
    executable location = /usr/bin/ansible-playbook
    python version = 2.7.5 (default, Sep 12 2018, 05:31:16) [GCC 4.8.5 20150623 (Red
    Hat 4.8.5-36)]
Using /home/student/troubleshoot-playbook/ansible.cfg as config file
setting up inventory plugins
Parsed /home/student/troubleshoot-playbook/inventory inventory source with ini
  plugin
Loading callback plugin default of type stdout, v2.0 from /usr/lib/python2.7/site-
packages/ansible/plugins/callback/default.pyc

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.example.com> ESTABLISH SSH CONNECTION FOR USER: devops
...output omitted...
fatal: [servera.lab.example.com]: UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: OpenSSH_7.4p1, OpenSSL 1.0.2k-
fips 26 Jan 2017\r\ndebug1: Reading configuration data /home/student/.ssh/config
\r\ndebug1: /home/student/.ssh/config line 1: Applying options for *\r\ndebug1:
  Reading configuration data /etc/ssh/ssh_config\r\ndebug1: /etc/ssh/ssh_config
  line 58: Applying options for *\r\ndebug1: auto-mux: Trying existing master\r
\ndebug1: Control socket \"/home/student/.ansible/cp/d4775f48c9\" does not exist\r
\ndebug2: resolving \"servera.lab.example.com\" port 22\r\nssh: Could not resolve
  hostname servera.lab.example.com: Name or service not known\r\n",
    "unreachable": true
}
```

```
...output omitted...
PLAY RECAP *****
servera.lab.exampple.com : ok=0     changed=0     unreachable=1    failed=0
```

- 15. Lorsque le niveau de détail maximal est utilisé avec Ansible, pour vérifier le résultat, mieux vaut utiliser le fichier journal Ansible que la console. Vérifiez le résultat de la commande précédente dans le fichier **/home/student/troubleshoot-playbook/ansible.log**.

```
[student@workstation troubleshoot-playbook]$ tail ansible.log
2018-12-17 19:22:50,508 p=18287 u=student | task path: /home/student/
troubleshoot-playbook/samba.yml:2
2018-12-17 19:22:50,549 p=18287 u=student | fatal: [servera.lab.exampple.com]:
UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: OpenSSH_7.4p1, OpenSSL 1.0.2k-
fips 26 Jan 2017\r\ndebug1: Reading configuration data /home/student/.ssh/config
\r\ndebug1: /home/student/.ssh/config line 1: Applying options for *\r\ndebug1:
Reading configuration data /etc/ssh/ssh_config\r\ndebug1: /etc/ssh/ssh_config
line 58: Applying options for *\r\ndebug1: auto-mux: Trying existing master\r
\ndebug1: Control socket \"/home/student/.ansible/cp/d4775f48c9\" does not exist\r
\ndebug2: resolving \"servera.lab.exampple.com\" port 22\r\nssh: Could not resolve
hostname servera.lab.exampple.com: Name or service not known\r\n",
    "unreachable": true
}
2018-12-17 19:22:50,550 p=18287 u=student | to retry, use: --limit @/home/
student/troubleshoot-playbook/samba.retry

2018-12-17 19:22:50,550 p=18287 u=student | PLAY RECAP *****
2018-12-17 19:22:50,550 p=18287 u=student | servera.lab.exampple.com : ok=0
changed=0     unreachable=1    failed=0
```

- 16. Parcourez le fichier **inventory** à la recherche d'erreurs. Notez que le groupe **[samba\_servers]** a été mal orthographié **servera.lab.example.com**. Corrigez cette erreur comme indiqué ci-dessous :

```
...output omitted...
[samba_servers]
servera.lab.example.com
...output omitted...
```

- 17. Exécutez de nouveau le playbook. La tâche **debug install\_state variable** renvoie le message **The state for the samba service is installed**. Cette tâche utilise le module **debug** et affiche la valeur de la variable **install\_state**. Une erreur est également présente dans la tâche **deliver samba config**, car aucun fichier **samba.j2** n'est disponible dans le répertoire de travail **/home/student/troubleshoot-playbook/**.

```
[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. Modifiez le playbook, et corrigez le paramètre **src** dans la tâche *deliver samba config* en spécifiant la valeur **samba.conf.j2**. Lorsque vous avez terminé, le playbook doit se présenter comme suit :

```

...output omitted...
- name: deliver samba config
  template:
    src: samba.conf.j2
    dest: /etc/samba/smb.conf
    owner: root
...output omitted...

```

- 19. Exécutez de nouveau le playbook. Exécutez le playbook en utilisant l'option **--step**. L'exécution doit se dérouler sans erreur.

```

[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

```

## Nettoyage

À partir de workstation, exécutez le script **lab troubleshoot-playbook cleanup** pour effacer cet exercice.

```
[student@workstation ~]$ lab troubleshoot-playbook cleanup
```

L'exercice guidé est maintenant terminé.

# RÉSOLUTION DES PROBLÈMES LIÉS AUX HÔTES GÉRÉS ANSIBLE

---

## OBJECTIFS

Après avoir terminé cette section, les stagiaires doivent pouvoir résoudre les échecs sur les hôtes gérés lors de l'exécution d'un playbook.

## UTILISATION DU MODE CHECK EN TANT QU'OUTIL DE TEST

Vous pouvez utiliser la commande **ansible-playbook --check** pour réaliser des tests de base sur un playbook. Cette option exécute le playbook sans modifier la configuration des hôtes gérés. Si un module utilisé dans le playbook prend en charge le *mode Check*, les modifications apportées aux hôtes gérés apparaissent mais ne sont pas exécutées. Si le mode Check n'est pas pris en charge par un module, les modifications ne sont pas affichées mais le module n'effectue toujours aucune action.

```
[student@demo ~]$ ansible-playbook --check playbook.yml
```



### NOTE

Si les tâches contiennent des conditions, il est possible que la commande **ansible-playbook --check** ne fonctionne pas correctement.

Vous pouvez également contrôler si des tâches individuelles sont exécutées en mode Check avec le paramètre `check_mode`. Si `check_mode: yes` a été défini pour une tâche, elle fonctionne toujours en mode check, que vous ayez ou non transmis l'option `--check` à **ansible-playbook**. De même, si `check_mode: no` est défini pour une tâche, elle fonctionne toujours normalement, même si vous transmettez `--check` à **ansible-playbook**.

La tâche suivante est toujours exécutée en mode Check et n'apporte aucune modification.

```
tasks:
  - name: task always in check mode
    shell: uname -a
    check_mode: yes
```

La tâche suivante est toujours exécutée normalement, même lorsqu'elle est démarrée avec **ansible-playbook --check**.

```
tasks:
  - name: task always runs even in check mode
    shell: uname -a
    check_mode: no
```

Cela peut être utile, car vous pouvez exécuter normalement la majeure partie d'un playbook tout en testant des tâches individuelles avec `check_mode: yes`. De même, les tests exécutés en

mode Check sont susceptibles de fournir des résultats raisonnables en exécutant certaines tâches qui rassemblent des faits ou définissent des variables pour les conditions, mais ne modifient pas les hôtes gérés avec **check\_mode: no**.

Une tâche peut déterminer si le playbook est en cours d'exécution en mode Check, en testant la valeur de la variable magique `ansible_check_mode`. Cette variable booléenne est définie sur `true` si le playbook est en cours d'exécution en mode Check.



### MISE EN GARDE

Si `check_mode: no` est défini pour des tâches, elles fonctionneront même lorsque le playbook est exécuté avec `ansible-playbook --check`. Par conséquent, vous ne pouvez pas compter sur le fait que l'option `--check` n'apportera aucune modification aux hôtes gérés, sans confirmation préalable en inspectant le playbook et les rôles ou tâches qui lui sont associés.



### NOTE

Si vous avez des playbooks plus anciens qui utilisent `always_run: yes` pour forcer les tâches à s'exécuter normalement même en mode Check, vous devrez remplacer ce code par `check_mode: no` dans Ansible 2.6 et version ultérieure.

La commande `ansible-playbook` fournit également une option `--diff`. Cette option signale les modifications apportées aux fichiers de modèle sur les hôtes gérés. Si elle est combinée à l'option `--check`, ces modifications s'affichent mais ne sont pas effectuées.

```
[student@demo ~]$ ansible-playbook --check --diff playbook.yml
```

## MODULES DE TEST

Certains modules permettent d'obtenir des informations supplémentaires sur le statut d'un hôte géré. La liste suivante inclut une partie des modules Ansible pouvant être utilisés pour tester et résoudre des problèmes sur les hôtes gérés.

- Le module `uri` permet de vérifier si une API RESTful renvoie le contenu attendu.

```
tasks:  
  - uri:  
      url: http://api.myapp.com  
      return_content: yes  
      register: apiresponse  
  
  - fail:  
      msg: 'version was not provided'  
      when: "'version' not in apiresponse.content"
```

- Le module `script` prend en charge l'exécution d'un script sur des hôtes gérés et échoue si le code retourné par ce script est différent de zéro. Le script doit exister sur le nœud de contrôle et est transféré et exécuté sur les hôtes gérés.

```
tasks:
```

```
- script: check_free_memory
```

- Le module **stat** rassemble des faits pour un fichier similaire à la commande **stat**. Vous pouvez l'utiliser pour enregistrer une variable, puis effectuer un test pour déterminer si le fichier existe ou pour obtenir d'autres informations sur le fichier. Si le fichier n'existe pas, la tâche **stat** n'échouera pas, mais sa variable enregistrée renverra **false** pour `*.stat.exists`.

Dans cet exemple, une application est toujours en cours d'exécution si `/var/run/app.lock` existe, auquel cas le play doit s'interrompre.

```
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: not lock.stat.exists
```

- Vous pouvez utiliser le module **assert** à la place du module **fail**. Le module **assert** prend en charge une option `that` qui prend une liste de conditions. Si l'une de ces conditions est fausse, la tâche échoue. Vous pouvez utiliser les options `success_msg` et `fail_msg` pour personnaliser le message à imprimer s'il signale une réussite ou un échec.

L'exemple suivant répète le précédent, mais utilise **assert** à la place de **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
```

## RÉSOLUTION DES PROBLÈMES LIÉS AUX CONNEXIONS

De nombreux problèmes courants lors de l'utilisation d'Ansible pour gérer des hôtes sont associés à des connexions à l'hôte et à des problèmes de configuration liés à l'utilisateur distant et à l'augmentation des priviléges.

Si vous rencontrez des problèmes d'authentification sur un hôte géré, assurez-vous que `remote_user` est correctement défini dans votre fichier de configuration ou dans votre play. Vous devez également vérifier que les clés SSH appropriées ont été configurées ou que le mot de passe fourni est correct pour cet utilisateur.

Vérifiez que `become` est correctement réglé et que vous utilisez le bon `become_user` (`root` par défaut). Vous devez confirmer que vous entrez le bon mot de passe `sudo` et que `sudo` sur l'hôte géré est configuré correctement.

Un problème plus subtil concerne les paramètres d'inventaire. Pour un serveur complexe avec plusieurs adresses réseau, vous devrez peut-être utiliser une adresse ou un nom DNS particulier lors de la connexion à ce système. Vous souhaiterez peut-être ne pas utiliser cette adresse comme nom d'inventaire de la machine pour une meilleure lisibilité. Vous pouvez définir une variable d'inventaire d'hôtes, `ansible_host`, qui remplacera le nom de l'inventaire par un nom ou une adresse IP différents et sera utilisée par Ansible pour se connecter à cet hôte. Cette variable peut être définie dans le fichier `host_vars` ou répertoire de cet hôte, ou peut être défini dans le fichier d'inventaire lui-même.

Par exemple, l'entrée d'inventaire suivante configure Ansible de sorte qu'il se connecte à 192.0.2.4 lors du traitement de l'hôte `web4.phx.example.com`:

```
web4.phx.example.com ansible_host=192.0.2.4
```

C'est un moyen utile de contrôler la manière dont Ansible se connecte aux hôtes gérés. Cependant, cela peut aussi entraîner des problèmes si la valeur de `ansible_host` est incorrecte.

## UTILISATION DE COMMANDES AD HOC POUR LES TESTS

Les exemples suivants illustrent certaines des vérifications pouvant être effectuées sur un hôte géré en utilisant des commandes ad hoc.

Vous avez utilisé le module `ping` pour vérifier si vous pouvez vous connecter à des hôtes gérés. En fonction des options que vous transmettez, vous pouvez également l'utiliser pour vérifier si l'augmentation des privilèges et les informations d'identification sont correctement configurées.

```
[student@demo ~]$ ansible demohost -m ping
demohost | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
[student@demo ~]$ ansible demohost -m ping --become
demohost | FAILED! => {
    "changed": false,
    "module_stderr": "sudo: a password is required\n",
    "module_stdout": "",
    "msg": "MODULE FAILURE\nSee stdout/stderr for the exact error",
    "rc": 1
}
```

L'exemple suivant indique l'espace disponible sur les disques configurés sur l'hôte géré `demohost`. Cela peut être utile pour confirmer que le système de fichiers sur l'hôte géré n'est pas plein.

```
[student@demo ~]$ ansible demohost -m command -a 'df'
```

L'exemple suivant indique la mémoire libre disponible sur l'hôte géré `demohost`.

```
[student@demo ~]$ ansible demohost -m command -a 'free -m'
```

## NIVEAU DE TEST CORRECT

Ansible est conçu pour s'assurer que la configuration incluse dans les playbooks et réalisée par les modules est correcte. Il surveille tous les échecs signalés par les modules et arrête instantanément le playbook au moindre échec rencontré. L'administrateur a ainsi la garantie que chaque tâche effectuée avant un échec ne contient aucune erreur.

Grâce à ce processus, il est généralement inutile de vérifier si une tâche gérée par Ansible a été correctement appliquée sur les hôtes gérés. Il est judicieux d'ajouter des vérifications d'intégrité aux playbooks, ou de les exécuter directement sous la forme de commandes ad hoc, lorsqu'une résolution de problèmes plus directe est nécessaire. Toutefois, vous devez limiter la complexité de vos tâches et de vos plays afin de revérifier les tests effectués par les modules eux-mêmes.



### RÉFÉRENCES

#### **Mode Check (« Essai à blanc ») -- Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/user\\_guide/playbooks\\_checkmode.html](https://docs.ansible.com/ansible/2.7/user_guide/playbooks_checkmode.html)

#### **Stratégies de test -- Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/reference\\_appendices/test\\_strategies.html](https://docs.ansible.com/ansible/2.7/reference_appendices/test_strategies.html)

## ► EXERCICE GUIDÉ

# RÉSOLUTION DES PROBLÈMES LIÉS AUX HÔTES GÉRÉS ANSIBLE

Dans cet exercice, vous allez résoudre les échecs liés aux tâches survenant sur l'un de vos hôtes gérés lors de l'exécution d'un playbook.

## RÉSULTATS

Vous devez pouvoir dépanner des hôtes gérés.

Connectez-vous à `workstation` en tant qu'utilisateur `student` avec le mot de passe `student`.

Sur `workstation`, exéutez le script `lab troubleshoot-host setup`. Ce dernier vérifie qu'Ansible est installé sur `workstation` et télécharge les fichiers `inventory`, `mailrelay.yml` et `postfix-relay-main.conf.j2` à partir de `http://materials.example.com/troubleshoot-host/` dans le répertoire `/home/student/troubleshoot-host/`.

```
[student@workstation ~]$ lab troubleshoot-host setup
```

- ▶ 1. Sur `workstation`, accédez au répertoire `/home/student/troubleshoot-host/`.

```
[student@workstation ~]$ cd ~/troubleshoot-host/
```

- ▶ 2. Exécutez de nouveau le playbook `mailrelay.yml` en mode Check.

```
[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"}
...output omitted...
PLAY RECAP ****
```

```
servera.lab.example.com      : ok=6      changed=3      unreachable=0      failed=1
```

La tâche *verify main.cf file exists* utilise le module `stat` pour confirmer que **main.cf** existe sur `servera.lab.example.com`.

La tâche *email notification of always\_bcc config* a échoué et n'a donc pas reçu le résultat de la tâche *check for always\_bcc*, car le playbook a été exécuté en mode Check.

- 3. Utilisez une commande ad hoc pour vérifier l'en-tête du fichier **/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
```

La commande a échoué car le playbook était exécuté en mode Check. Postfix n'est pas installé sur `servera.lab.example.com`

- 4. Exécutez de nouveau le playbook, sans sélectionner le mode Check. L'erreur qui figurait dans la tâche *email notification of always\_bcc config* doit disparaître.

```
[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
```

- 5. À l'aide d'une commande ad hoc, affichez la partie supérieure du fichier **/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 | SUCCESS | rc=0 >>
# Ansible managed
#
# Global Postfix configuration file. This file lists only a subset
# 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/.
```

Une ligne apparaît contenant la chaîne « Ansible managed », confirmant que le fichier a été mis à jour et qu'il est désormais géré par Ansible.

- 6. Ajoutez une tâche pour activer le service `smtp` derrière le pare-feu.

```
[student@workstation troubleshoot-host]$ vim mailrelay.yml
...output omitted...
- name: postfix firewalld config
  firewalld:
    state: enabled
    permanent: true
    immediate: true
    service: smtp
...output omitted...
```

- 7. Exécutez le playbook. La tâche `postfix firewalld config` doit s'exécuter sans erreur.

```
[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
```

- 8. À l'aide d'une commande ad hoc, vérifiez que le service `smtp` est bien configuré sur le pare-feu sur `servera.lab.example.com`.

```
[student@workstation troubleshoot-host]$ ansible servera.lab.example.com \
> -u devops -b -a "firewall-cmd --list-services"
servera.lab.example.com | CHANGED | rc=0 >>
dhcpv6-client samba smtp ssh
```

- 9. Utilisez `telnet` pour tester si le service SMTP écoute le port `TCP/25` sur `servera.lab.example.com`. Déconnectez-vous lorsque vous avez terminé.

```
[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.
```

## Nettoyage

À partir de workstation, exécutez le script **lab troubleshoot-host cleanup** pour effacer cet exercice.

```
[student@workstation ~]$ lab troubleshoot-host cleanup
```

L'exercice guidé est maintenant terminé.

## ► OPEN LAB

# RÉSOLUTION DES PROBLÈMES LIÉS À ANSIBLE

## LISTE DE CONTRÔLE DES PERFORMANCES

Au cours de cet atelier, vous allez résoudre les problèmes qui surviennent lorsque vous essayez d'exécuter un playbook qui vous a été fourni.

## RÉSULTATS

Vous devez pouvoir :

- Résoudre des problèmes liés aux playbooks.
- Résoudre des problèmes liés aux hôtes gérés.

Connectez-vous à `workstation` en tant qu'utilisateur `student` avec le mot de passe `student`. Exécutez la commande `lab troubleshoot-review setup`.

```
[student@workstation ~]$ lab troubleshoot-review setup
```

Ce script vérifie si Ansible est installé sur `workstation`. Il crée le répertoire `~/troubleshoot-review/`, puis le sous-répertoire `html`. Il télécharge à partir de `http://materials.example.com/labs/troubleshoot-review/` les fichiers `ansible.cfg`, `inventory-lab`, `secure-web.yml` et `vhosts.conf` dans le répertoire `/home/student/troubleshoot-review/` ainsi que le fichier `index.html` dans le répertoire `/home/student/troubleshoot-review/html/`.

1. À partir du répertoire `~/troubleshoot-review`, vérifiez la syntaxe du playbook `secure-web.yml`. Ce playbook contient un play qui configure Apache HTTPD avec TLS/SSL pour les hôtes du groupe `webservers`. Corrigez le problème signalé.
2. Vérifiez de nouveau la syntaxe du playbook `secure-web.yml`. Corrigez le problème signalé.
3. Vérifiez la syntaxe du playbook `secure-web.yml` une troisième fois. Corrigez le problème signalé.
4. Vérifiez la syntaxe du playbook `secure-web.yml` une quatrième fois. Il ne devrait rester aucune erreur de syntaxe.
5. Exécutez le playbook `secure-web.yml`. Ansible ne parvient pas à se connecter à `serverb.lab.example.com`. Corrigez ce problème.
6. Exécutez de nouveau le playbook `secure-web.yml`. Ansible n'est pas en mesure de s'authentifier en tant qu'utilisateur distant `devops` sur l'hôte géré. Corrigez ce problème.
7. Exécutez le playbook `secure-web.yml` une troisième fois. Corrigez le problème signalé.
8. Exécutez le playbook `secure-web.yml` une fois de plus. Il doit correctement s'exécuter. Utilisez une commande ad hoc pour vérifier que le service `httpd` est en cours d'exécution.

## Évaluation

À partir de workstation, exécutez le script **lab troubleshoot-review grade** pour confirmer que l'exercice est réussi.

```
[student@workstation troubleshoot-review]$ lab troubleshoot-review grade
```

## Nettoyage

À partir de workstation, exécutez le script **lab troubleshoot-review cleanup** pour effacer l'atelier.

```
[student@workstation troubleshoot-review]$ lab troubleshoot-review cleanup
```

## ► SOLUTION

# RÉSOLUTION DES PROBLÈMES LIÉS À ANSIBLE

## LISTE DE CONTRÔLE DES PERFORMANCES

Au cours de cet atelier, vous allez résoudre les problèmes qui surviennent lorsque vous essayez d'exécuter un playbook qui vous a été fourni.

## RÉSULTATS

Vous devez pouvoir :

- Résoudre des problèmes liés aux playbooks.
- Résoudre des problèmes liés aux hôtes gérés.

Connectez-vous à `workstation` en tant qu'utilisateur `student` avec le mot de passe `student`. Exécutez la commande `lab troubleshoot-review setup`.

```
[student@workstation ~]$ lab troubleshoot-review setup
```

Ce script vérifie si Ansible est installé sur `workstation`. Il crée le répertoire `~student/troubleshoot-review/`, puis le sous-répertoire `html`. Il télécharge à partir de `http://materials.example.com/labs/troubleshoot-review/` les fichiers `ansible.cfg`, `inventory-lab`, `secure-web.yml` et `vhosts.conf` dans le répertoire `/home/student/troubleshoot-review/` ainsi que le fichier `index.html` dans le répertoire `/home/student/troubleshoot-review/html/`.

1. À partir du répertoire `~/troubleshoot-review`, vérifiez la syntaxe du playbook `secure-web.yml`. Ce playbook contient un play qui configure Apache HTTPD avec TLS/SSL pour les hôtes du groupe `webservers`. Corrigez le problème signalé.
  - 1.1. Sur `workstation`, accédez au répertoire de projet `/home/student/troubleshoot-review`.

```
[student@workstation ~]$ cd ~/troubleshoot-review/
```

- 1.2. Vérifiez la syntaxe du playbook `secure-web.yml`. Si tout se passe correctement, ce playbook configure Apache HTTPD avec TLS/SSL pour les hôtes du groupe `webservers`.

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

```
ERROR! Syntax Error while loading YAML.
mapping values are not allowed in this context
```

```
The error appears to have been in '/home/student/Ansible-course/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. Corrigez le problème de syntaxe dans la définition de la variable `random_var` en ajoutant des guillemets droits à la chaîne `This is colon: test`. La modification qui en résulte doit s'afficher comme suit :

```
...output omitted...  
vars:  
  random_var: "This is colon: test"  
...output omitted...
```

2. Vérifiez de nouveau la syntaxe du playbook **secure-web.yml**. Corrigez le problème signalé.
  - 2.1. Vérifiez de nouveau la syntaxe de **secure-web.yml** à l'aide de **ansible-playbook --syntax-check**.

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

```
ERROR! Syntax Error while loading YAML.  
      did not find expected '-' indicator
```

```
The error appears to have been in '/home/student/Ansible-course/troubleshoot-review/secure-web.yml': line 43, 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. Corrigez tous les problèmes de syntaxe dans la mise en retrait. Supprimez l'espace superflu au début des éléments de tâche `start and enable web services`. La modification qui en résulte doit s'afficher comme suit :

```
...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  
    tags:
```

```

    - services

    - name: deliver content
      copy:
        dest: /var/www/vhosts/serverb-secure
        src: html/
...output omitted...

```

3. Vérifiez la syntaxe du playbook **secure-web.yml** une troisième fois. Corrigez le problème signalé.

3.1. Vérifiez la syntaxe du playbook **secure-web.yml**.

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

```
ERROR! Syntax Error while loading YAML.
  found unacceptable key (unhashable type: 'AnsibleMapping')
```

The error appears to have been in '/home/student/Ansible-course/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. Corrigez la variable **item** dans la tâche **install web server packages**. Ajoutez des guillemets droits à **{{ item }}**. La modification qui en résulte doit s'afficher comme suit :

```

...output omitted...
    - name: install web server packages
      yum:
        name: "{{ item }}"
        state: latest
      notify:
        - restart services
      tags:
        - packages
      loop:
```

```
- httpd
- mod_ssl
- crypto-utils
...output omitted...
```

4. Vérifiez la syntaxe du playbook **secure-web.yml** une quatrième fois. Il ne devrait rester aucune erreur de syntaxe.

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

playbook: secure-web.yml
```

5. Exécutez le playbook **secure-web.yml**. Ansible ne parvient pas à se connecter à `serverb.lab.example.com`. Corrigez ce problème.

- 5.1. Exécutez le playbook **secure-web.yml**. Cela échoue et produit une erreur.

```
[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: Warning: Permanently added
'serverc.lab.example.com,172.25.250.12' (ECDSA) to the list of known hosts.\r
\nPermission denied (publickey,gssapi-keyex,gssapi-with-mic,password).\r\n",
"unreachable": true}
      to retry, use: --limit @/home/student/troubleshoot-review/secure-web.retry

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

- 5.2. Exécutez de nouveau le playbook **secure-web.yml**, en ajoutant le paramètre **-vvvv** pour augmenter le niveau de détail de la sortie.  
Notez qu'Ansible semble se connecter à `serverc.lab.example.com` au lieu de `serverb.lab.example.com`.

```
[student@workstation troubleshoot-review]$ ansible-playbook secure-web.yml -vvvv
...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 -vvv -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. Corrigez la ligne dans le fichier **inventory-lab**. Supprimez la variable d'hôte `ansible_host` de sorte que le fichier apparaisse comme indiqué ci-dessous :

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

6. Exécutez de nouveau le playbook **secure-web.yml**. Ansible n'est pas en mesure de s'authentifier en tant qu'utilisateur distant devops sur l'hôte géré. Corrigez ce problème.

- 6.1. Exécutez le playbook **secure-web.yml**.

```
[student@workstation troubleshoot-review]$ ansible-playbook secure-web.yml -vvvv
...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
<serverb.lab.example.com> EXEC ssh -C -vvv -o ControlMaster=auto
-o ControlPersist=60s -o Port=22 -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/ansible-ssh-%C -tt
serverb.lab.example.com '/bin/sh -c """( umask 22 && mkdir -p `"
echo $HOME/.ansible/tmp/ansible-tmp-1460241127.16-3182613343880 `" &&
echo "` echo $HOME/.ansible/tmp/ansible-tmp-1460241127.16-3182613343880
`" )"""""
...output omitted...
fatal: [serverb.lab.example.com]: UNREACHABLE! => {
...output omitted...
```

- 6.2. Modifiez le playbook **secure-web.yml** pour vous assurer que `devops` est le `remote_user` pour le play. Les premières lignes du playbook doivent se présenter comme suit :

```
---
# start of secure web server playbook
- name: create secure web service
  hosts: webservers
  remote_user: devops
...output omitted...
```

7. Exécutez le playbook **secure-web.yml** une troisième fois. Corrigez le problème signalé.

- 7.1. Exécutez le playbook **secure-web.yml**.

```
[student@workstation troubleshoot-review]$ ansible-playbook secure-web.yml -vvvv
...output omitted...
failed: [serverb.lab.example.com] => (item=[u'httpd', u'mod_ssl',
u'crypto-utils']) => {"changed": true, "failed": true, "invocation": {
"module_args": {"conf_file": null, "disable_gpg_check": false, "disablerepo": null,
"enablerepo": null, "exclude": null, "install_repoquery": true,
"list": null, "name": ["httpd", "mod_ssl", "crypto-utils"], "state": "latest",
```

```
"update_cache": false}, "module_name": "yum"}, "item": ["httpd", "mod_ssl", "crypto-utils"], "msg": "You need to be root to perform this command.\n", "rc": 1, "results": ["Loaded plugins: langpacks, search-disabled-repos\n"]}
```

...output omitted...

- 7.2. Modifiez le play pour vous assurer qu'il est défini sur **become: true** ou **become: yes**. La modification qui en résulte doit s'afficher comme suit :

```
---
# start of secure web server playbook
- name: create secure web service
  hosts: webservers
  remote_user: devops
  become: true
  ...output omitted...
```

8. Exécutez le playbook **secure-web.yml** une fois de plus. Il doit correctement s'exécuter. Utilisez une commande ad hoc pour vérifier que le service `httpd` est en cours d'exécution.

- 8.1. Exécutez le playbook **secure-web.yml**.

```
[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=[u'httpd', u'mod_ssl', u'crypto-utils'])
...output omitted...
TASK [httpd_conf_syntax variable] ****
ok: [serverb.lab.example.com] => {
    "msg": "The httpd_conf_syntax variable value is {'stderr_lines': [u'Syntax OK'], u'changed': True, u'end': u'2018-12-17 23:31:53.191871', 'failed': False, u'stdout': u'', u'cmd': [u'/sbin/httpd', u'-t'], u'rc': 0, u'start': u'2018-12-17 23:31:53.149759', u'stderr': u'Syntax OK', u'delta': u'0:00:00.042112', 'stdout_lines': [], 'failed_when_result': False}"
}
...output omitted...
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. Utilisez une commande ad hoc pour déterminer l'état du service `httpd` sur `serverb.lab.example.com`. À présent, le service `httpd` doit être exécuté sur `serverb.lab.example.com`.

```
[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 Tue 2016-05-03 19:43:39 CEST; 12s ago  
...output omitted...
```

## Évaluation

À partir de workstation, exécutez le script **lab troubleshoot-review grade** pour confirmer que l'exercice est réussi.

```
[student@workstation troubleshoot-review]$ lab troubleshoot-review grade
```

## Nettoyage

À partir de workstation, exécutez le script **lab troubleshoot-review cleanup** pour effacer l'atelier.

```
[student@workstation troubleshoot-review]$ lab troubleshoot-review cleanup
```

# RÉSUMÉ

---

Dans ce chapitre, vous avez appris les principes suivants :

- Ansible contient une fonction de journalisation intégrée. Cette fonction n'est pas activée par défaut.
- Le paramètre `log_path` dans la section **default** du fichier de configuration **ansible.cfg** spécifie l'emplacement du fichier journal vers lequel toutes les sorties Ansible sont redirigées.
- Tout en exécutant un playbook, le module `debug` fournit des informations de débogage supplémentaires (comme la valeur actuelle d'une variable).
- L'option `-v` de la commande **ansible-playbook** fournit plusieurs niveaux de détail applicables aux résultats. Cela est utile pour le débogage de tâches Ansible lors de l'exécution d'un playbook.
- L'option `--check` permet aux modules Ansible prenant en charge le mode Check d'afficher les modifications apportées, au lieu de les appliquer aux hôtes gérés.
- Des vérifications supplémentaires peuvent être effectuées sur les hôtes gérés à l'aide de commandes ad hoc.
- Si le playbook se termine correctement, il est inutile de vérifier une nouvelle fois la configuration appliquée par Ansible.



## CHAPITRE 10

# AUTOMATISATION DES TÂCHES D'ADMINISTRATION LINUX

### PROJET

Automatiser les tâches d'administration courantes de systèmes Linux avec Ansible.

### OBJECTIFS

- S'abonner aux systèmes, configurer les canaux de logiciels et les référentiels, et gérer les paquetages RPM sur des hôtes gérés.
- Gérer les utilisateurs et les groupes Linux, configurer SSH et modifier la configuration de Sudo sur les hôtes gérés.
- Gérer le démarrage du service, planifier les processus avec at, cron et systemd, redémarrer et contrôler la cible de démarrage par défaut sur les hôtes gérés.
- Partitionner les périphériques de stockage, configurer LVM, formater les partitions ou les volumes logiques, monter les systèmes de fichiers et ajouter des fichiers ou des espaces d'échange.

### SECTIONS

- Introduction à Automatisation des tâches d'administration Linux
- Gestion des logiciels et des abonnements (exercice guidé)
- Gestion des utilisateurs et de l'authentification (et exercice guidé)
- Gestion du processus de démarrage et des processus planifiés (exercice guidé)
- Gestion du stockage (exercice guidé)

# PRÉSENTATION DE L'AUTOMATISATION DES TÂCHES D'ADMINISTRATION LINUX

---

## OBJECTIFS

Après avoir terminé cette section, les stagiaires doivent pouvoir expliquer l'objet de ce chapitre.

## PRÉSENTATION DE L'EXERCICE PRATIQUE

Ce chapitre consiste en une série d'exercices pratiques guidés visant à illustrer l'automatisation de certaines tâches courantes d'administration de système Linux à l'aide de Red Hat Ansible Engine. Cette section présente un bref aperçu des prochains exercices.

## GESTION DES LOGICIELS ET DES ABONNEMENTS

Le premier exercice examinera en détail les tâches de gestion de paquetage sur Red Hat Enterprise Linux. Vous savez déjà comment installer et mettre à jour les paquetages RPM à l'aide des modules yum ou package. Dans cet exercice, vous apprendrez à utiliser certains modules Ansible supplémentaires :

- yum\_repository pour configurer un référentiel YUM tiers ;
- rpm\_key pour installer et gérer les clés de signature RPM afin que le système puisse vérifier l'authenticité des paquetages téléchargés sans intervention humaine ;
- package\_facts pour collecter des faits supplémentaires contenant des informations détaillées sur les paquetages RPM installés sur vos hôtes gérés.

Il existe deux autres modules que vous devriez connaître et que nous ne couvrons pas actuellement dans l'atelier : redhat\_subscription et rhsm\_repository.

Le module redhat\_subscription utilise la commande **subscription-manager** pour enregistrer vos hôtes gérés dans Red Hat Subscription Management (RHSM) ou sur un serveur Red Hat Satellite 6. Vous pouvez utiliser des clés d'activation, assigner automatiquement des abonnements ou spécifier un pool d'abonnements spécifique à utiliser, et configurer les paramètres de proxy.

Le module rhsm\_repository utilise **subscription-manager** pour activer ou désactiver des référentiels RHSM ou Satellite spécifiques pour un serveur enregistré. Il prend un identifiant de référentiel ou une liste d'ID de référentiels à vérifier et un état **enabled** ou **disabled**.

```
- name: Ansible 2.7 repository is enabled on control node
  rhsm_repository:
    name: rhel-7-server-ansible-2.7-rpms
    state: enabled
```

## GESTION DES UTILISATEURS ET DE L'AUTHENTIFICATION

Cet exercice examine certaines approches pour gérer les utilisateurs locaux :

- Utilisez le module `users` et le module `groups` pour vous assurer que les utilisateurs et les groupes locaux existent sur vos nœuds gérés et disposent de paramètres cohérents.
- Utilisez le module `authorized_key` pour vous assurer que l'authentification via une clé SSH est configurée pour différents utilisateurs. Il a recours à une fonctionnalité avancée appelée *lookups* que nous n'avons pas encore abordée dans ce cours, mais il existe également d'autres méthodes pour déployer des clés à l'aide de cette fonctionnalité.
- Vérifiez que `sshd` est configuré pour désactiver les connexions root, en utilisant le module `lineinfile` et en redémarrant le démon avec le module `service` si `lineinfile` effectue une modification.
- Utilisez le module `lineinfile` pour configurer un groupe avec des priviléges `sudo` en éditant `/etc/sudoers`. (Une autre façon de résoudre ce problème aurait été d'utiliser le module `copy` ou `file` pour s'assurer qu'un fichier approprié configurant `sudo` existe dans le répertoire `/etc/sudoers.d`.)

Si vous utilisez Red Hat Identity Management (FreeIPA) dans votre environnement, vous voudrez peut-être consulter la vaste collection de modules `ipa_*` disponibles avec Ansible.

## GESTION DU PROCESSUS DE DÉMARRAGE ET DES PROCESSUS PLANIFIÉS

Dans cet exercice, vous examinez différentes manières de planifier des processus à exécuter ultérieurement et d'ajuster le processus de démarrage. Vous allez :

- utiliser le module `cron` pour planifier des tâches répétitives en gérant des fichiers `/etc/cron.d` et leur contenu ;
- utiliser le module `at` pour programmer une tâche ponctuelle qui sera effectuée ultérieurement sur vos hôtes gérés ;
- ajuster la cible de démarrage `systemd` par défaut en utilisant le module `file` pour vous assurer que le lien symbolique défini par `systemctl set-default` pointe vers la bonne cible ;
- redémarrer les systèmes avec le module `reboot`.

Si vous gérez des travaux récurrents avec les minuteurs `systemd`, vous pouvez utiliser le module `service` permettant de gérer ces unités de la même manière que les unités de service :

```
- name: Keep temporary directories clean
  service:
    name: systemd-tmpfiles-clean.timer
    enabled: yes
    state: started
```

## GESTION DU STOCKAGE

Cet exercice utilise Ansible pour gérer les partitions de disque, les volumes logiques, les systèmes de fichiers et le montage du système de fichiers. Vous allez :

- utiliser le module `parted` pour vous assurer qu'un disque est correctement partitionné ;
- utiliser le module `lvg` pour gérer les groupes de volumes et les volumes physiques LVM ;
- utiliser le module `lvol` pour gérer les volumes logiques LVM ;

- utiliser le fait `ansible_facts['lvm']` pour obtenir des informations sur les paramètres LVM existants sur les hôtes gérés ;
- utiliser le module `filesystem` pour formater les périphériques de bloc avec des systèmes de fichiers ;
- utiliser le module `mount` pour ajouter des entrées pour vos systèmes de fichiers dans `/etc/fstab` et les monter immédiatement.



### MISE EN GARDE

Agissez toujours avec précaution lorsque vous testez des plays Ansible qui modifient le stockage. Les erreurs que vous faites dans votre playbook risquent de supprimer ou de reformater par inadvertance les systèmes de fichiers.

Il est donc très utile que vous vous familiarisiez avec ces modules sur un système de test. Assurez-vous également que les playbooks utilisant ces modules se comportent de manière idempotente et sécurisée, en les exécutant une seconde fois sur le système de test.

Les anciennes versions d'Ansible ne comprenaient pas le module `parted`. Une astuce plus ancienne pour éviter de partitionner un disque qui contenait déjà des partitions consistait à utiliser le module `command` pour exécuter `parted` directement, mais uniquement si le fichier de périphérique de la partition n'existant pas déjà :

```
- name: /dev/sdb1 is partitioned
  command: >
    parted --script /dev/sdb mklabel gpt mkpart primary 1MiB 100%
  args:
    creates: /dev/sdb1
```

Le fait `ansible_facts['devices']` contient également de nombreuses variables utiles pour les conditions lors de la création et du formatage du stockage, et peut également être utilisé pour gérer plus soigneusement les systèmes de fichiers et le stockage sur des serveurs Linux.



### RÉFÉRENCES

#### Modules de paquetage

[https://docs.ansible.com/ansible/latest/modules/list\\_of\\_packaging\\_modules.html](https://docs.ansible.com/ansible/latest/modules/list_of_packaging_modules.html)

#### Modules système

[https://docs.ansible.com/ansible/latest/modules/list\\_of\\_system\\_modules.html](https://docs.ansible.com/ansible/latest/modules/list_of_system_modules.html)

#### Modules d'identité

[https://docs.ansible.com/ansible/latest/modules/list\\_of\\_identity\\_modules.html](https://docs.ansible.com/ansible/latest/modules/list_of_identity_modules.html)

## ► EXERCICE GUIDÉ

# GESTION DES LOGICIELS ET DES ABONNEMENTS

Dans cet exercice, vous allez configurer un nouveau référentiel YUM et installer les paquetages sur vos hôtes gérés.

## RÉSULTATS

Vous devez pouvoir :

- Utiliser le module `yum_repository` pour configurer un référentiel YUM.
- Utiliser le module `rpm_key` pour gérer les clés GPG RPM.
- Utiliser le module `package_facts` pour obtenir des informations sur les paquetages installés sur un hôte.

## Présentation du scénario

Votre organisation exige que tous les hôtes disposent du paquetage `example-motd` installé. Ce paquetage est fourni par un référentiel YUM interne géré par votre organisation pour héberger des paquetages logiciels développés en interne.

Vous êtes chargé d'écrire un playbook pour vous assurer que le paquetage `example-motd` est installé sur l'hôte distant. Le playbook doit assurer la configuration du référentiel YUM interne.

Le référentiel est situé sur `http://materials.example.com/yum/repository`. Tous les paquetages RPM sont signés avec une paire de clés GPG organisationnelle. La clé publique GPG est disponible à l'adresse `http://materials.example.com/yum/repository/RPM-GPG-KEY-example`.

Sur `workstation`, exécutez le script de configuration de l'atelier pour vérifier que l'environnement est prêt pour le début de l'atelier. Le script crée le répertoire de travail **system-software** et le renseigne à l'aide d'un fichier de configuration Ansible, d'un inventaire d'hôtes et de fichiers d'atelier.

```
[student@workstation ~]$ lab system-software setup
```

- 1. En tant qu'utilisateur `student` sur `workstation`, accédez au répertoire de travail `/home/student/system-software`.

```
[student@workstation ~]$ cd ~/system-software
[student@workstation system-software]$
```

- 2. Commencez à écrire le playbook `repo_playbook.yml`. Définissez un seul play dans le playbook qui cible tous les hôtes. Ajoutez une clause `vars` qui définit une variable unique `custom_pkg` avec la valeur `example-motd`. Ajoutez la clause `tasks` au playbook.

Le playbook contient maintenant :

```
---
- name: Repository Configuration
  hosts: all
  vars:
    custom_pkg: example-motd
  tasks:
```

► 3. Ajoutez les deux tâches au playbook.

Utilisez le module `package_facts` dans la première tâche pour rassembler des informations sur les paquetages installés sur l'hôte distant. Cette tâche remplit le fait `ansible_facts.packages`.

Utilisez le module `debug` dans la deuxième tâche pour imprimer la version installée du paquetage référencé par la variable `custom_pkg`. N'exécutez cette tâche que si le paquetage personnalisé est trouvé dans le fait `ansible_facts.packages`.

Exécutez le playbook `repo_playbook.yml`.

- 3.1. Ajoutez la première tâche au playbook. Configurez le mot-clé `manager` du module `package_facts` avec la valeur `auto`. La première tâche contient les éléments suivants :

```
- name: Gather Package Facts
  package_facts:
    manager: auto
```

- 3.2. Ajoutez une deuxième tâche au playbook qui utilise le module `debug` pour afficher la valeur de la variable `ansible_facts.packages[custom_pkg]`. Ajoutez une clause `when` à la tâche pour vérifier si la valeur de la variable `custom_pkg` est contenue dans la variable `ansible_facts.packages`. La deuxième tâche contient les éléments suivants :

```
- name: Show Package Facts for the custom package
  debug:
    var: ansible_facts.packages[custom_pkg]
  when: custom_pkg in ansible_facts.packages
```

- 3.3. Exécutez le playbook :

```
[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
```

La tâche de débogage est ignorée car le paquetage *example-motd* n'est pas installé sur l'hôte distant.

- 4. Ajoutez une troisième tâche qui utilise le module `yum_repository` pour garantir la configuration du référentiel YUM interne sur l'hôte distant. Veillez à ce que les exigences suivantes soient respectées :

- La configuration du référentiel est stockée dans le fichier `/etc/yum.repos.d/example.repo`
- L'ID de référentiel est **example-internal**
- L'URL de base est `http://material.example.com/yum/repository`
- Le référentiel est configuré pour vérifier les signatures GPG RPM
- La description du référentiel est **Example Inc. Internal YUM repo**

La troisième tâche contient les éléments suivants :

```
- 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. Ajoutez une quatrième tâche au play qui utilise le module `rpm_key` pour s'assurer que la clé publique du référentiel est présente sur l'hôte distant. L'URL de la clé publique du référentiel est `http://materials.example.com/yum/repository/RPM-GPG-KEY-example`.

La quatrième tâche apparaît comme suit :

```
- name: Ensure Repo RPM Key is Installed
rpm_key:
  key: http://materials.example.com/yum/repository/RPM-GPG-KEY-example
  state: present
```

- 6. Ajoutez une cinquième tâche pour vous assurer que le paquetage référencé par la variable `custom_pkg` est installé sur l'hôte distant.

La cinquième tâche apparaît comme suit :

```
- name: Install Example motd package
yum:
  name: "{{ custom_pkg }}"
  state: present
```

- 7. Le fait `ansible_facts.packages` n'est pas mis à jour lorsqu'un nouveau paquetage est installé sur un hôte distant.

Copiez la deuxième tâche et ajoutez-la en tant que sixième tâche dans le play. Exécutez le playbook et vérifiez que le fait `ansible_facts.packages` ne contient aucune information sur le paquetage `example-motd` installé sur l'hôte distant.

7.1. La sixième tâche contient une copie de la deuxième tâche :

```
- name: Show Package Facts for the custom package
  debug:
    var: ansible_facts.packages[custom_pkg]
  when: custom_pkg in ansible_facts.packages
```

Le playbook complet se présente maintenant comme suit :

```
---
- 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. Exécutez le playbook.

```
[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
```

- ① La tâche **Gather Package Facts** détermine les données contenues dans le fait `ansible_facts.packages`.
- ② La tâche est ignorée car le paquetage `example-motd` est installé après la tâche **Gather Package Facts**.

- 8. Insérez une tâche immédiatement après la tâche **Install Example motd package** en utilisant le module `package_facts` pour mettre à jour les faits du paquetage. Définissez le mot-clé `manager` du module avec la valeur `auto`.

Le playbook complet est présenté ci-dessous :

```
---
- 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. Utilisez une commande ad hoc Ansible pour supprimer le paquetage *example-motd* installé lors de l'exécution précédente du playbook. Exécutez le playbook avec la tâche `package_facts` insérée et utilisez la sortie pour vérifier l'installation du paquetage *example-motd*.

- 9.1. Pour supprimer le paquetage *example-motd* de tous les hôtes, utilisez la commande **ansible all** avec les options **-m yum** et **-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 => {
  "ansible_facts": {
    "pkg_mgr": "yum"
  },
  "changed": true,
  "msg": "",
  "rc": 0,
  "results": [
...output omitted...
]
```

}

## 9.2. Exédez le playbook.

```
[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
```

- ① Aucun fait de paquetage n'existe pour le paquetage `example-motd`, car le paquetage n'est pas installé sur l'hôte distant.
- ② Le paquetage `example-motd` est installé à la suite de cette tâche, comme indiqué par le statut **changed**.
- ③ Cette tâche met à jour les faits de paquetage avec des informations sur le paquetage `example-motd`.
- ④ Le fait du paquetage `example-motd` existe et indique qu'un seul paquetage `example-motd` est installé. La version du paquetage installé est **1.0**.

## Nettoyage

Sur workstation, exéutez le script **lab system-software cleanup** pour effacer les ressources créées au cours de cet exercice.

```
[student@workstation ~]$ lab system-software cleanup
```

L'exercice guidé est maintenant terminé.



### RÉFÉRENCES

**yum\_repository : ajouter ou supprimer des référentiels YUM – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/modules/yum\\_repository\\_module.html](https://docs.ansible.com/ansible/2.7/modules/yum_repository_module.html)

**rpm\_key : ajouter ou supprimer une clé gpg de la base de données rpm – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/modules/rpm\\_key\\_module.html](https://docs.ansible.com/ansible/2.7/modules/rpm_key_module.html)

**package\_facts : informations sur le paquetage en tant que faits – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/modules/package\\_facts\\_module.html](https://docs.ansible.com/ansible/2.7/modules/package_facts_module.html)

## ► EXERCICE GUIDÉ

# GESTION DES UTILISATEURS ET DE L'AUTHENTIFICATION

Dans cet exercice, vous allez créer plusieurs utilisateurs sur vos hôtes gérés et renseigner les clés SSH autorisées pour ces derniers.

## RÉSULTATS

Vous devez pouvoir :

- Créer un groupe d'utilisateurs.
- Utiliser le module `user` pour gérer des utilisateurs.
- Utiliser le module `authorized_key` pour renseigner les clés autorisées SSH.
- Utiliser le module `lineinfile` pour modifier les fichiers `sudoers` et `sshd_config`.

## Présentation du scénario

Votre organisation nécessite que tous les hôtes disposent des mêmes utilisateurs locaux. Ces utilisateurs doivent appartenir au groupe d'utilisateurs `webadmin` qui a la capacité d'utiliser la commande `sudo` sans spécifier de mot de passe. De plus, les clés publiques SSH des utilisateurs doivent être distribuées dans l'environnement, et l'utilisateur `root` ne doit pas être autorisé à se connecter directement à l'aide de SSH.

Vous êtes chargé d'écrire un playbook pour vous assurer que les utilisateurs et le groupe d'utilisateurs sont présents sur l'hôte distant. Le playbook doit garantir que les utilisateurs peuvent se connecter à l'aide de la clé SSH autorisée, ainsi qu'utiliser `sudo` sans spécifier de mot de passe, et que l'utilisateur `root` ne peut pas se connecter directement à l'aide de SSH.

Sur `workstation`, exécutez le script de configuration de l'atelier pour vérifier que l'environnement est prêt pour le début de l'atelier. Le script crée le répertoire de travail `system-users` et le renseigne à l'aide d'un fichier de configuration Ansible, d'un inventaire d'hôtes et de fichiers d'atelier.

```
[student@workstation ~]$ lab system-users setup
```

- 1. En tant qu'utilisateur `student` sur `workstation`, accédez au répertoire de travail `/home/student/system-users`.

```
[student@workstation ~]$ cd ~/system-users
[student@workstation system-users]$
```

- 2. Consultez le fichier de variable `vars/users_vars.yml` existant.

```
[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
```

Il utilise le nom de variable `username` pour définir le nom d'utilisateur correct, et la variable `groups` pour définir des groupes supplémentaires auxquels l'utilisateur devrait appartenir.

- 3. Commencez à écrire le playbook `users.yml`. Définissez un seul play dans le playbook qui cible le groupe d'hôtes `webservers`. Ajoutez une clause `vars_files` définissant l'emplacement du nom de fichier `vars/users_vars.yml` qui a été créé pour vous et qui contient tous les noms d'utilisateur requis pour cet exercice. Ajoutez la clause `tasks` au playbook.

Utilisez un éditeur de texte pour créer le playbook `users.yml`. Le playbook doit contenir les éléments suivants :

```
---
- name: Create multiple local users
  hosts: webservers
  vars_files:
    - vars/users_vars.yml
  tasks:
```

- 4. Ajoutez les deux tâches au playbook.

Utilisez le module `group` dans la première tâche pour créer le groupe d'utilisateurs `webadmin` sur l'hôte distant. Cette tâche crée le groupe `webadmin`.

Utilisez le module `user` dans la deuxième tâche pour créer les utilisateurs du fichier `vars/users_vars.yml`.

Exécutez le playbook `users.yml`.

- 4.1. Ajoutez la première tâche au playbook. La première tâche contient les éléments suivants :

```
- name: Add webadmin group
  group:
    name: webadmin
    state: present
```

- 4.2. Ajoutez une deuxième tâche au playbook qui utilise le module `user` pour créer les utilisateurs. Ajoutez une clause `loop: "{{ users }}"` à la tâche pour parcourir le fichier de variable pour chaque nom d'utilisateur trouvé dans le fichier `vars/users_vars.yml`. Comme nom (`name:`) d'utilisateurs, utilisez le nom de variable `item.username`. De cette façon, le fichier de variable peut contenir des informations supplémentaires pouvant être utiles pour créer les utilisateurs, telles

que les groupes auxquels les utilisateurs doivent appartenir. La deuxième tâche contient les éléments suivants :

```
- name: Create user accounts
  user:
    name: "{{ item.username }}"
    groups: "webadmin"
  loop: "{{ users }}"
```

#### 4.3. Exécutez le playbook :

```
[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. Ajoutez une troisième tâche qui utilise le module `authorized_key` pour s'assurer que les clés publiques SSH ont été correctement distribuées sur l'hôte distant. Dans le répertoire **files**, chaque utilisateur possède un fichier de clé publique SSH unique. Le module parcourt la liste des utilisateurs, trouve la clé appropriée en utilisant la variable `username` et transmet la clé à l'hôte distant.

La troisième tâche contient les éléments suivants :

```
- name: Add authorized keys
  authorized_key:
    user: "{{ item.username }}"
    key: "{{ lookup('file', 'files/' + item.username + '.key.pub') }}"
  loop: "{{ users }}"
```

- 6. Ajoutez une quatrième tâche au play qui utilise le module `lineinfile` pour modifier le fichier de configuration `sudo` et permettre aux membres du groupe `webadmin` d'utiliser `sudo` sans mot de passe sur l'hôte distant.

La quatrième tâche apparaît comme suit :

```
- name: Modify sudo config to allow webadmin users sudo without a password
  lineinfile:
    dest: "/etc/sudoers"
    state: "present"
    regexp: "^%webadmin"
    line: "%webadmin ALL=(ALL) NOPASSWD: ALL"
```

- 7. Ajoutez une cinquième tâche pour vous assurer que l'utilisateur `root` n'est pas autorisé à se connecter directement à l'aide de SSH. Utilisez `notify: "Restart sshd"` afin de déclencher un gestionnaire pour redémarrer SSH.

La cinquième tâche apparaît comme suit :

```
- name: Disable root login via SSH
  lineinfile:
    dest: "/etc/ssh/sshd_config"
    regexp: "^PermitRootLogin"
    line: "PermitRootLogin no"
  notify: "Restart sshd"
```

- 8. Dans la première ligne après l'emplacement du fichier de variable, ajoutez une nouvelle définition de gestionnaire. Nommez-la **Restart sshd**.

8.1. Le gestionnaire doit être défini comme suit :

```
...output omitted...
- vars/users_vars.yml
handlers:
- name: "Restart sshd"
  service:
    name: "sshd"
    state: "restarted"
```

Le playbook complet se présente maintenant comme suit :

```
---
- 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: "webadmin"
  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
  lineinfile:
    dest: "/etc/sudoers"
    state: "present"
    regexp: "^%webadmin"
    line: "%webadmin ALL=(ALL) NOPASSWD: ALL"

- name: Disable root login via SSH
  lineinfile:
    dest: "/etc/ssh/sshd_config"
    regexp: "^\s*PermitRootLogin\s+"
    line: "PermitRootLogin no"
  notify: "Restart sshd"

```

## 8.2. Exécutez le playbook.

```

[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': 'user3', 'groups': 'webadmin'})
ok: [servera.lab.example.com] => (item={'username': 'user4', 'groups': 'webadmin'})

```

```

ok: [servera.lab.example.com] => (item={'username': 'user5', 'groups': 'webadmin'})

TASK [Add authorized keys] ****
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'})

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. Comme l'utilisateur **user1**, connectez-vous au serveur **servera** en utilisant SSH. Une fois connecté, utilisez la commande **sudo su -** pour prendre l'identité de l'utilisateur **root**.

9.1. Utilisez SSH en tant qu'utilisateur **user1** et connectez-vous au serveur **servera**.

```
[student@workstation system-users]$ ssh user1@servera
[user1@servera ~]$
```

9.2. Basculez vers l'utilisateur **root**.

```
[user1@servera ~]$ sudo su -
Last login: Wed Dec 19 05:39:53 EST 2018 on pts/0
root@servera ~]#
```

9.3. Déconnectez-vous du serveur **servera**.

```
[root@servera ~]$ exit
logout
[user1@servera ~]$ exit
logout
Connection to servera closed.
[student@workstation system-users]$
```

- 10. Essayez de vous connecter au serveur **servera**, en tant qu'utilisateur **root** directement. Cette étape doit échouer car la configuration SSH a été modifiée pour ne pas autoriser la connexion directe en tant qu'utilisateur **root**.

- 10.1. À partir de **workstation**, utilisez SSH en tant que **root** pour vous connecter au serveur **servera**.

```
[student@workstation system-users]$ ssh root@servera
root@servera's password: redhat
Permission denied, please try again.
root@servera's password:
```

Cela confirme que la configuration SSH a refusé l'accès direct au système pour l'utilisateur **root**.

## Nettoyage

Sur **workstation**, exécutez le script **lab system-users cleanup** pour effacer les ressources créées au cours de cet exercice.

```
[student@workstation ~]$ lab system-users cleanup
```

L'exercice guidé est maintenant terminé.



### RÉFÉRENCES

**authorized\_key : ajoute ou supprime une clé autorisée SSH – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/modules/authorized\\_key\\_module.html](https://docs.ansible.com/ansible/2.7/modules/authorized_key_module.html)

## ► EXERCICE GUIDÉ

# GESTION DU PROCESSUS DE DÉMARRAGE ET DES PROCESSUS PLANIFIÉS

Dans cet exercice, vous allez gérer le processus de démarrage, planifier des tâches récurrentes et redémarrer des hôtes gérés.

## RÉSULTATS

Vous devez pouvoir utiliser un playbook pour :

- planifier une tâche **cron** récurrente ;
- supprimer un seule tâche **cron** spécifique d'un fichier **crontab** ;
- planifier une tâche **at** ;
- définir la cible de démarrage par défaut sur les hôtes gérés ;
- redémarrer les hôtes gérés.

À partir de **workstation**, exécutez le script **lab system-process setup** afin de configurer l'environnement pour cet exercice. Le script crée le répertoire de projet **system-process**, et télécharge le fichier de configuration Ansible ainsi que le fichier d'inventaire d'hôtes nécessaire pour cet exercice.

```
[student@workstation ~]$ lab system-process setup
```

- 1. En tant qu'utilisateur **student** sur **workstation**, accédez au répertoire de travail **/home/student/system-process**.

```
[student@workstation ~]$ cd ~/system-process
[student@workstation system-process]$
```

- 2. Créez un playbook, **create\_crontab\_file.yml**, dans le répertoire de travail actuel. Configurez le playbook afin qu'il utilise le module **cron** pour créer le fichier **crontab /etc/cron.d/add-date-time** qui planifie une tâche cron récurrente. La tâche doit être exécutée par l'utilisateur **devops** toutes les deux minutes entre **09:00** et **16:59** du **lundi** au **vendredi**. Elle doit ajouter la date et l'heure actuelles au fichier **/home/devops/my\_datetime\_cron\_job**
- 2.1. Créez un playbook, **create\_crontab\_file.yml**, et ajoutez-y les lignes nécessaires pour démarrer le play. Il doit cibler les hôtes gérés **webservers** et activer l'augmentation des privilèges.

---

```
- name: Recurring cron job
hosts: webservers
become: true
```

- 2.2. Définissez une tâche qui utilise le module **cron** pour planifier une tâche cron récurrente.

**NOTE**

Le module **cron** fournit une option **name** afin de décrire de manière unique l'entrée de fichier crontab et de garantir les résultats escomptés. La description est ajoutée au fichier crontab. Par exemple, l'option **name** est requise si vous supprimez une entrée crontab à l'aide de **state=absent**. En outre, lorsque l'état par défaut **state=present** est défini, l'option **name** empêche la création constante d'une nouvelle entrée crontab, quels que soient les éléments existants.

```
tasks:
- name: Crontab file exists
  cron:
    name: Add date and time to a file
```

- 2.3. Configurez la tâche afin qu'elle soit exécutée toutes les deux minutes entre **09:00** et **16:59** du **lundi** au **vendredi**.

```
minute: "*/2"
hour: 9-16
day: 1-5
```

- 2.4. Utilisez le paramètre **cron\_file** afin d'utiliser le fichier crontab **/etc/cron.d/add-date-time** à la place du fichier crontab d'un utilisateur individuel **/var/spool/cron/**. Un chemin relatif placera le fichier dans le répertoire **/etc/cron.d**. Si le paramètre **cron\_file** est utilisé, vous devez également spécifier le paramètre **user**.

```
user: devops
job: date >> /home/devops/my_date_time_cron_job
cron_file: add-date-time
state: present
```

- 2.5. Quand vous avez terminé, le playbook doit se présenter comme suit. Vérifiez si le playbook est correct.

```
---
- 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
day: 1-5
user: devops
job: date >> /home/devops/my_date_time_cron_job
cron_file: add-date-time
state: present
```

- 2.6. Vérifiez la syntaxe du playbook en exécutant la commande **ansible-playbook --syntax-check create\_crontab\_file.yml**. En cas d'erreurs, corrigez-les avant de passer à l'étape suivante.

```
[student@workstation system-process]$ ansible-playbook --syntax-check \
> create_crontab_file.yml

playbook: create_crontab_file.yml
```

- 2.7. Exécutez le playbook.

```
[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. Exécutez une commande ad hoc pour vérifier que le fichier cron **/etc/cron.d/add-date-time** existe et que son contenu est correct.

```
[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. Créez un playbook, **remove\_cron\_job.yml**, dans le répertoire de travail actuel. Configurez le playbook afin qu'il utilise le module cron pour supprimer la tâche cron **Add date and time to a file** du fichier crontab **/etc/cron.d/add-date-time**.

- 3.1. Créez un playbook, **remove\_cron\_job.yml**, et ajoutez les lignes suivantes :

```
---
- name: Remove scheduled cron job
  hosts: webservers
  become: true
```

```
tasks:
  - name: Cron job removed
    cron:
      name: Add date and time to a file
      cron_file: add-date-time
      state: absent
```

- 3.2. Vérifiez la syntaxe du playbook en exécutant la commande **ansible-playbook --syntax-check remove\_cron\_job.yml**. En cas d'erreurs, corrigez-les avant de passer à l'étape suivante.

```
[student@workstation system-process]$ ansible-playbook --syntax-check \
> remove_cron_job.yml

playbook: remove_cron_job.yml
```

- 3.3. Exécutez le playbook.

```
[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. Exécutez une commande ad hoc pour vérifier que le fichier cron **/etc/cron.d/add-date-time** continue d'exister tandis que la tâche cron a été supprimée.

```
[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. Créez un playbook, **schedule\_at\_task.yml**, dans le répertoire de travail actuel. Configurez le playbook afin qu'il utilise le module **at** pour planifier une tâche qui sera exécutée ultérieurement pendant une minute. La tâche doit exécuter la commande **date** et rediriger sa sortie vers le fichier **/home/devops/my\_at\_date\_time**. Utilisez l'option **unique: yes** pour empêcher l'ajout d'une nouvelle tâche si la commande existe déjà dans la file d'attente **at**.

- 4.1. Créez un playbook, **schedule\_at\_task.yml**, et ajoutez les lignes suivantes :

```
---
- 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"
      count: 1
      units: minutes
      unique: yes
      state: present

```

- 4.2. Vérifiez la syntaxe du playbook en exécutant la commande **ansible-playbook --syntax-check schedule\_at\_task.yml**. En cas d'erreurs, corrigez-les avant de passer à l'étape suivante.

```
[student@workstation system-process]$ ansible-playbook --syntax-check \
> schedule_at_task.yml

playbook: schedule_at_task.yml
```

- 4.3. Exécutez le playbook.

```
[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. Après avoir patienté pendant une minute, le temps que la commande **at** ait été exécutée, exécutez des commandes ad hoc pour vérifier que le fichier **/home/devops/my\_at\_date\_time** existe et que son contenu est correct.

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "ls -l my_at_date_time"
servera.lab.example.com | CHANGED | rc=0 >>
-rw-rw-r--. 1 devops devops 29 Dec 19 00:14 my_at_date_time

[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "cat my_at_date_time"
servera.lab.example.com | CHANGED | rc=0 >>
Wed Dec 19 00:14:00 CST 2018
```

- 5. Créez un playbook, **set\_default\_boot\_target\_graphical.yml**, dans le répertoire de travail actuel. Configurez le playbook afin qu'il utilise le module **file** pour changer le

lien symbolique sur les hôtes gérés afin de référencer la cible de démarrage `graphical-target`.

### **NOTE**

Dans le module `file`, la valeur du paramètre `src` est ce à quoi le lien symbolique fait référence. La valeur du paramètre `dest` est le lien symbolique.

- 5.1. Créez un playbook, `set_default_boot_target_graphical.yml`, et ajoutez les lignes suivantes :

```
---
- 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. Vérifiez la syntaxe du playbook en exécutant la commande `ansible-playbook --syntax-check set_default_boot_target_graphical.yml`. En cas d'erreurs, corrigez-les avant de passer à l'étape suivante.

```
[student@workstation system-process]$ ansible-playbook --syntax-check \
> set_default_boot_target_graphical.yml

playbook: set_default_boot_target_graphical.yml
```

- 5.3. Avant d'exécuter le playbook, exécutez une commande ad hoc pour vérifier que la cible de démarrage par défaut est `multi-user.target` :

```
[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. Exécutez le playbook.

```
[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. Exécutez une commande ad hoc pour vérifier que la cible de démarrage par défaut est maintenant `graphical.target`.

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "systemctl get-default"
servera.lab.example.com | CHANGED | rc=0 >>
graphical.target
```

- 6. Créez un playbook, `reboot_hosts.yml`, dans le répertoire de travail actuel qui redémarre les hôtes gérés. Il n'est pas nécessaire de redémarrer un serveur après avoir modifié la cible par défaut. Toutefois, savoir comment créer un playbook qui redémarre des hôtes gérés peut s'avérer utile.

- 6.1. Créez un playbook, `reboot_hosts.yml`, et ajoutez les lignes suivantes :

```
---
- name: Reboot hosts
  hosts: webservers
  become: true

  tasks:
    - name: Hosts are rebooted
      reboot:
```

- 6.2. Vérifiez la syntaxe du playbook en exécutant la commande `ansible-playbook --syntax-check reboot_hosts.yml`. En cas d'erreurs, corrigez-les avant de passer à l'étape suivante.

```
[student@workstation system-process]$ ansible-playbook --syntax-check \
> reboot_hosts.yml
```

```
playbook: reboot_hosts.yml
```

- 6.3. Avant d'exécuter le playbook, exécutez une commande ad hoc pour déterminer l'horodatage du dernier redémarrage du système.

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "who -b"
servera.lab.example.com | CHANGED | rc=0 >>
system boot 2018-12-19 17:13
```

- 6.4. Exécutez le playbook.

```
[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. Exécutez une commande ad hoc pour déterminer l'horodatage du dernier redémarrage du système. L'horodatage affiché après l'exécution du playbook devrait être postérieur.

```

[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "who -b"
servera.lab.example.com | CHANGED | rc=0 >>
      system boot  2018-12-19 21:06

```

- 6.6. Exécutez une deuxième commande ad hoc pour déterminer que la cible de démarrage `graphical.target` a survécu au redémarrage.

```

[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "systemctl get-default"
servera.lab.example.com | CHANGED | rc=0 >>
graphical.target

```

- ▶ 7. Pour maintenir la cohérence tout au long des exercices restants, rétablissez la configuration initiale de la cible de démarrage par défaut, `multi-user.target`. Créez un playbook, `set_default_boot_target_multi-user.yml`, dans le répertoire de travail actuel. Configurez le playbook afin qu'il utilise le module `file` pour changer le lien symbolique sur les hôtes gérés afin de référencer la cible de démarrage `multi-user.target`.

- 7.1. Créez un playbook, `set_default_boot_target_multi-user.yml`, et ajoutez les lignes suivantes :

```

---
- 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. Vérifiez la syntaxe du playbook en exécutant la commande **ansible-playbook --syntax-check set\_default\_boot\_target\_multi-user.yml**. En cas d'erreurs, corrigez-les avant de passer à l'étape suivante.

```
[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. Exécutez le playbook.

```
[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. Exécutez une commande ad hoc pour vérifier que la cible de démarrage par défaut est maintenant **multi-user.target**.

```
[student@workstation system-process]$ ansible webservers -u devops -b \
> -a "systemctl get-default"
servera.lab.example.com | CHANGED | rc=0 >>
multi-user.target
```

## Nettoyage

À partir de **workstation**, exécutez le script **lab system-process cleanup** pour effacer cet exercice.

```
[student@workstation ~]$ lab system-process cleanup
```

L'exercice guidé est maintenant terminé.



## RÉFÉRENCES

**at : planifier l'exécution d'une commande ou d'un fichier script via la commande at – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/modules/at\\_module.html](https://docs.ansible.com/ansible/2.7/modules/at_module.html)

**cron : gérer les entrées cron.d et crontab – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/modules/cron\\_module.html](https://docs.ansible.com/ansible/2.7/modules/cron_module.html)

**reboot : redémarrer une machine – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/modules/reboot\\_module.html](https://docs.ansible.com/ansible/2.7/modules/reboot_module.html)

## ► EXERCICE GUIDÉ

# GESTION DU STOCKAGE

Dans cet exercice, vous allez partitionner un nouveau disque, créer des volumes logiques et les formater avec des systèmes de fichiers XFS, puis les monter immédiatement et automatiquement au démarrage sur vos hôtes gérés.

## RÉSULTATS

Vous devez pouvoir :

- Utiliser le module `parted` pour configurer les partitions de périphérique de bloc.
- Utiliser le module `lvg` pour gérer les groupes de volumes LVM.
- Utiliser le module `lvol` pour gérer les volumes logiques LVM.
- Utiliser le module `filesystem` pour créer les systèmes de fichiers.
- Utiliser le module `mount` pour contrôler et configurer les points de montage dans `/etc/fstab`.

À partir de `workstation`, exécutez le script `lab system-storage setup` afin de configurer l'environnement pour cet exercice. Ce script crée le répertoire de projet `system-storage`, et télécharge le fichier de configuration Ansible ainsi que le fichier d'inventaire des hôtes nécessaire pour cet exercice.

```
[student@workstation ~]$ lab system-storage setup
```

## Présentation du scénario

Vous êtes responsable de la gestion d'un ensemble de serveurs Web. Une pratique recommandée pour la configuration du serveur Web consiste à stocker les données du serveur Web sur une partition ou un volume logique distinct.

Vous écrirez un playbook pour :

- gérer les partitions du périphérique `/dev/vdb` ;
- gérer un groupe de volumes nommé `apache-vg` pour les données du serveur Web ;
- créer deux volumes logiques nommés `content-lv` et `logs-lv`, tous deux appuyés par le groupe de volumes `apache-vg` ;
- créer un système de fichiers XFS sur les deux volumes logiques ;
- monter le volume logique `content-lv` sur `/var/www` ;
- monter le volume logique `logs-lv` sur `/var/log/httpd`.

Si les exigences de stockage du serveur Web changent, mettez à jour les variables du playbook appropriées et réexécutez le playbook. Le playbook doit être idempotent.

- 1. En tant qu'utilisateur student sur workstation, accédez au répertoire de travail **/home/student/system-storage**.

```
[student@workstation ~]$ cd ~/system-storage
[student@workstation system-storage]$
```

- 2. Examinez le fichier playbook squelette **storage.yml** et le fichier de variables associé **storage\_vars.yml** dans le répertoire du projet. Exécutez le playbook.

2.1. Examinez le 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 }}"
```

Le nom de chaque tâche agit comme un aperçu de la procédure prévue à mettre en œuvre. Dans les étapes suivantes, vous mettrez à jour et modifierez ces six tâches.

## 2.2. Examinez le fichier de variables **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
```

Ce fichier décrit la structure prévue des partitions, des groupes de volumes et des volumes logiques sur chaque serveur Web. La première partition commence avec un décalage de 1 Mio du début du périphérique /dev/vdb, et se termine avec un décalage de 257 Mio, pour une taille totale de 256 Mio.

Chaque serveur Web est doté d'un groupe de volumes, nommé **apache-vg**, contenant la première partition du périphérique /dev/vdb.

Chaque serveur Web dispose de deux volumes logiques. Le premier volume logique est nommé **content-lv**, d'une taille de 64 Mio, attaché au groupe de volumes **apache-vg**, et monté sur **/var/www**. Le second volume logique est nommé **logs-lv**, d'une taille de 128 Mio, attaché au groupe de volumes **apache-vg**, et monté sur **/var/log/httpd**.



### NOTE

Le groupe de volumes **apache-vg** a une capacité de 256 Mio, car il est appuyé par la partition /dev/vdb1. Il fournit une capacité suffisante pour les deux volumes logiques.

## 2.3. Exécutez le playbook **storage.yml**.

```
[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={'start': u'1MiB', 'end': u'257MiB',
  'number': 1}) => {
    "msg": "TODO"
}

...output omitted...

TASK [Each Logical Volume is mounted] ****
ok: [servera.lab.example.com] => (item={'vgroup': u'apache-vg', 'size': u'64M',
  'mount_path': u'/var/www', 'name': u'content-lv'}) => {
    "msg": "TODO"
}
ok: [servera.lab.example.com] => (item={'vgroup': u'apache-vg', 'size': u'128M',
  'mount_path': u'/var/log/httpd', 'name': u'logs-lv'}) => {
    "msg": "TODO"
}

PLAY RECAP ****
servera.lab.example.com      : ok=7      changed=0      unreachable=0      failed=0

```

- ▶ 3. Modifiez la première tâche afin d'utiliser le module `parted` pour configurer une partition pour chaque élément de boucle. Chaque élément décrit une partition prévue du périphérique `/dev/vdb` sur chaque serveur Web :

#### **number**

Numéro de la partition, à utiliser comme valeur du mot-clé **number** pour le module `parted`.

#### **start**

Début de la partition, comme un décalage par rapport au début du périphérique de bloc. Utilisez ceci comme valeur du mot-clé **part\_start** pour le module `parted`.

#### **end**

Fin de la partition, comme un décalage par rapport au début du périphérique de bloc. Utilisez ceci comme valeur du mot-clé **part\_end** pour le module `parted`.

Voici le contenu de la première tâche :

```

- 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. Modifiez la deuxième tâche du play afin d'utiliser le module `lvg` pour configurer un groupe de volumes pour chaque élément de boucle. Chaque élément de la variable `volume_groups` décrit un groupe de volumes qui doit exister sur chaque serveur Web :

**name**

Nom du groupe de volumes, à utiliser comme valeur du mot-clé **vg** pour le module `lvg`.

**devices**

Liste des périphériques ou des partitions séparés par des virgules qui forment le groupe de volumes, à utiliser comme valeur du mot-clé **pvs** pour le module `lvg`.

Voici le contenu de la deuxième tâche :

```
- name: Ensure Volume Groups Exist
  lvg:
    vg: "{{ item.name }}"
    pvs: "{{ item.devices }}"
  loop: "{{ volume_groups }}"
```

- 5. Modifiez la troisième tâche du play afin d'utiliser le module `lvol` pour créer un volume logique pour chaque élément. Utilisez les mots-clés de l'élément pour créer le volume logique :

**name**

Nom du volume logique, à utiliser comme valeur du mot-clé **lv** pour le module `lvol`.

**vgroup**

Nom du groupe de volumes qui fournit le stockage pour le volume logique.

**size**

Taille du volume logique. La valeur de ce mot-clé est une valeur acceptable pour l'option **-L** de la commande `lvcreate`.

N'exécutez la tâche que si un volume logique n'existe pas déjà. Mettez à jour l'instruction **when** pour vérifier qu'un volume logique n'existe pas avec un nom correspondant à la valeur du mot-clé **name** de l'élément.

- 5.1. Modifiez la troisième tâche afin d'utiliser le module `lvol`. Définissez le nom du groupe de volumes, le nom du volume logique et la taille du volume logique à l'aide des mots-clés de chaque élément. Le contenu de la troisième tâche est maintenant :

```
- 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. Le fait Ansible `ansible_lvm` contient des informations sur les objets de gestion de volumes logiques sur chaque hôte. Utilisez une commande ad hoc pour voir le jeu actuel de volumes logiques sur l'hôte distant :

```
[student@workstation system-storage]$ ansible all -m setup -a \
```

```
> "filter=ansible_lvm"
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_lvm": {
            "lvs": {},
            "pvs": {},
            "vgs": {}
        }
    },
    "changed": false
}
```

La valeur du mot-clé `lvs` indique qu'il n'y a pas de volume logique sur l'hôte distant.

### 5.3. Exécutez le playbook pour créer les volumes logiques sur l'hôte distant.

```
[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. Exécutez une autre commande ad hoc pour voir la structure de la variable `ansible_lvm` lorsque des volumes logiques existent sur l'hôte distant.

```
[student@workstation system-storage]$ ansible all -m setup -a \
> "filter=ansible_lvm"
servera.lab.example.com | SUCCESS => {
    "ansible_facts": {
        "ansible_lvm": {
            "lvs": ①{
                "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 La valeur du mot-clé `lvs` est une structure de données de paire de clés. Les clés de cette structure sont les noms de tous les volumes logiques sur l'hôte. Cela indique que les deux volumes logiques `content-lv` et `logs-lv` existent. Pour chaque volume logique, le groupe de volumes correspondant est fourni par le mot-clé `vg`.
- ❷ Le mot-clé `pvs` contient des informations relatives aux volumes physiques sur l'hôte. Les informations indiquent que la partition `/dev/vdb1` appartient au groupe de volumes `apache-vg`.
- ❸ Le mot-clé `vgs` contient des informations relatives aux groupes de volumes sur l'hôte.

- 5.5. Mettez à jour l'instruction `when` pour vérifier qu'un volume logique n'existe pas avec un nom correspondant à la valeur du mot-clé `name` de l'élément. Le contenu de la troisième tâche est maintenant :

```

- 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. Modifiez la quatrième tâche afin d'utiliser le module `filesystem`. Configurez la tâche pour vous assurer que chaque volume logique est formaté en tant que système de fichiers XFS.

Rappelez-vous qu'un volume logique est associé au périphérique logique **/dev/<volume group name>/<logical volume name>**.

Voici le contenu de la quatrième tâche :

```
- name: Ensure XFS Filesystem exists on each LV
  filesystem:
    dev: "/dev/{{ item.vgroup }}/{{ item.name }}"
    fstype: xfs
    loop: "{{ logical_volumes }}"
```

- ▶ 7. Configurez la cinquième tâche pour vous assurer que chaque volume logique dispose de la capacité de stockage appropriée. Si la capacité du volume logique augmente, veillez à forcer l'extension du système de fichiers du volume.



#### MISE EN GARDE

Si la capacité d'un volume logique doit être réduite, cette tâche échouera car un système de fichiers XFS ne prend pas en charge la réduction de capacité.

Voici le contenu de la cinquième tâche :

```
- 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. Utilisez le module `mount` dans la sixième tâche pour vous assurer que chaque volume logique est monté sur le chemin de montage correspondant et persiste après un redémarrage.

Voici le contenu de la sixième tâche :

```
- name: Each Logical Volume is mounted
  mount:
    path: "{{ item.mount_path }}"
    src: "/dev/{{ item.vgroup }}/{{ item.name }}"
    fstype: xfs
    state: mounted
    loop: "{{ logical_volumes }}"
```

- ▶ 9. Examinez le playbook **storage.yml**. Exécutez le playbook et vérifiez que chaque volume logique est monté.

9.1. Examinez le 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
    lvol:
      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. Exécutez le playbook.

```
[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...})

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
```

Une tâche est ignorée lors de l'exécution car le playbook a déjà été exécuté avec les mêmes valeurs de variable. Il n'a pas été nécessaire de créer les volumes logiques.

9.3. Utilisez une commande ad hoc Ansible pour exécuter la commande **lsblk** sur l'hôte distant. La sortie indique les points de montage des volumes logiques.

```
[student@workstation system-storage]$ ansible all -a lsblk
servera.lab.example.com | CHANGED | rc=0 >>
NAME           MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
fd0            2:0     1   4K  0 disk
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. Augmentez la capacité du volume logique `content-lv` à 128 Mio, et le volume logique `logs-lv` à 256 Mio. Cela nécessite d'augmenter la capacité du groupe de volumes `apache-vg`.

Créez une partition d'une capacité de 256 Mio et ajoutez-la au groupe de volumes `apache-vg`.

- 10.1. Modifiez la définition de variable `partitions` dans le fichier `storage_vars.yml` pour ajouter une deuxième partition au périphérique `/dev/vdb`. Voici le contenu de la variable `partitions`:

```
partitions:
  - number: 1
    start: 1MiB
    end: 257MiB
  - number: 2
    start: 257MiB
    end: 513MiB
```

- 10.2. Copiez la définition de variable `volume_groups` dans le fichier `storage_vars.yml`. Ajoutez la deuxième partition à la liste des périphériques sauvegardant le groupe de volumes. Voici le contenu de la variable `volume_groups`:

```
volume_groups:
  - name: apache-vg
    devices: /dev/vdb1,/dev/vdb2
```

- 10.3. Doublez la capacité de chaque volume logique défini dans le fichier `storage_vars.yml`. Voici le contenu de la variable `logical_volumes`:

```
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. Exécutez le playbook. Vérifiez la nouvelle capacité de chaque volume logique.

```
[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'})

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...})

PLAY RECAP ****
servera.lab.example.com : ok=6    changed=3    unreachable=0    failed=0

```

La sortie indique les modifications apportées aux partitions et au groupe de volumes sur l'hôte distant et que les deux volumes logiques ont été redimensionnés.

- 10.5. Utilisez une commande ad hoc Ansible pour exécuter la commande **lsblk** sur l'hôte distant.

```

[student@workstation system-storage]$ ansible all -a lsblk
servera.lab.example.com | CHANGED | rc=0 >>
NAME          MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
fd0           2:0    1   4K  0 disk 
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

```

La sortie indique que chaque volume logique a la taille correcte et est monté sur le bon répertoire. Deux entrées existent pour chaque volume logique, car les fichiers

stockés sur le volume logique peuvent se trouver physiquement sur l'une des partitions (/dev/vdb1 ou /dev/vdb2).

## Nettoyage

Exécutez la commande **lab system-storage cleanup** pour nettoyer l'hôte géré.

```
[student@workstation ~]$ lab system-storage cleanup
```

L'exercice guidé est maintenant terminé.



### RÉFÉRENCES

**parted : configurer les partitions de périphérique de bloc – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/modules/parted\\_module.html](https://docs.ansible.com/ansible/2.7/modules/parted_module.html)

**lvg : configurer les groupes de volumes LVM – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/modules/lvg\\_module.html](https://docs.ansible.com/ansible/2.7/modules/lvg_module.html)

**lvol : configurer les volumes logiques LVM – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/modules/lvol\\_module.html](https://docs.ansible.com/ansible/2.7/modules/lvol_module.html)

**filesystem : créer un système de fichiers – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/modules/filesystem\\_module.html](https://docs.ansible.com/ansible/2.7/modules/filesystem_module.html)

**mount : contrôler les points de montage actifs et configurés – Documentation Ansible**

[https://docs.ansible.com/ansible/2.7/modules/mount\\_module.html](https://docs.ansible.com/ansible/2.7/modules/mount_module.html)

## CHAPITRE 11

# RÉVISION APPROFONDIE : AUTOMATION WITH ANSIBLE

### PROJET

Démontrer les compétences acquises dans ce cours en installant, en optimisant et en configurant Ansible pour la gestion des hôtes gérés.

### SECTIONS

- Révision complète

### ATELIERS

- Atelier : Déploiement d'Ansible
- Atelier : Création de playbooks
- Atelier : Création de rôles et utilisation d'inventaires dynamiques

# RÉVISION COMPLÈTE

---

## OBJECTIFS

À la fin de cette section, les stagiaires seront en mesure de mettre en pratique leurs connaissances et les compétences acquises dans *Automation with Ansible*.

## RÉVISION AUTOMATION WITH ANSIBLE

Avant de commencer la révision exhaustive de ce cours, les stagiaires doivent être familiarisés avec les rubriques abordées dans chaque chapitre.

Les stagiaires peuvent se référer aux précédentes sections du manuel pour en savoir plus.

### Chapitre 1, Présentation d'Ansible

Décrire les concepts Ansible et installer Red Hat Ansible Engine.

- Décrire les concepts, l'architecture et des cas d'utilisation d'Ansible.
- Installer Ansible sur un nœud de contrôle et décrire la distinction entre communauté Ansible et Red Hat Ansible Engine.

### Chapitre 2, Déploiement d'Ansible

Configurer Ansible pour gérer des hôtes et exécuter les commandes Ansible ad hoc.

- Décrire les concepts d'inventaire Ansible et gérer un fichier d'inventaire statique.
- Décrire où se trouvent les fichiers de configuration Ansible, comment Ansible les sélectionne, et les modifier pour appliquer les modifications aux paramètres par défaut.
- Exécuter une seule tâche d'automatisation Ansible à l'aide d'une commande ad hoc et expliquer certains cas d'utilisation de commandes ad hoc.

### Chapitre 3, Mise en œuvre de playbooks

Écrire un playbook Ansible simple et l'exécuter pour automatiser les tâches sur plusieurs hôtes.

- Écrire un playbook Ansible de base et l'exécuter en utilisant la commande **ansible-playbook**.
- Écrire un playbook qui utilise plusieurs plays et l'augmentation des privilèges par play.
- Utiliser efficacement **ansible-doc** pour apprendre à utiliser de nouveaux modules visant à implémenter des tâches pour un play.

### Chapitre 4, Gestion des variables et des faits

Rédigez des cahiers qui utilisent des variables et des faits pour simplifier la gestion du cahier et des faits afin de référencer des informations sur les hôtes gérés.

- Créer et référencer des variables qui affectent des hôtes ou groupes d'hôtes particuliers, le play ou l'environnement global, et décrire le fonctionnement de la priorité des variables.

- Chiffrer les variables sensibles à l'aide d'Ansible Vault et exécuter des playbooks faisant référence à des fichiers de variables chiffrées par Vault.
- Référencer les données sur les hôtes gérés à l'aide de faits Ansible et configurer des faits personnalisés sur des hôtes gérés.

## **Chapitre 5, Mise en œuvre d'un contrôle de tâche**

Gérer le contrôle de tâche, les gestionnaires et les erreurs de tâche dans les playbooks Ansible.

- Utiliser des boucles pour écrire des tâches efficaces et des conditions pour contrôler quand exécuter des tâches.
- Implémenter une tâche qui s'exécute uniquement lorsqu'une autre tâche modifie l'hôte géré.
- Contrôler ce qui se passe lorsqu'une tâche échoue et quelles conditions provoquent son échec.

## **Chapitre 6, Déploiement de fichiers sur des hôtes gérés**

Déployer, gérer et ajuster les fichiers sur des hôtes gérés par Ansible.

- Créer, installer, modifier et supprimer des fichiers sur des hôtes gérés, et gérer les autorisations, la propriété, le contexte SELinux et d'autres caractéristiques de ces fichiers.
- Déployer des fichiers sur des hôtes gérés personnalisés à l'aide de modèles Jinja2.

## **Chapitre 7, Gestion de projets volumineux**

Rédiger des playbooks optimisés pour des projets plus grands et plus complexes.

- Écrire des modèles d'hôte sophistiqués pour sélectionner efficacement les hôtes d'une commande play ou ad hoc.
- Décrire ce que sont les inventaires dynamiques, et installer et utiliser un script existant en tant que source d'inventaire dynamique Ansible.
- Régler le nombre de connexions simultanées ouvertes par Ansible sur les hôtes gérés et déterminer comment Ansible traite les groupes d'hôtes gérés via les tâches du play.
- Gérer des playbooks volumineux en important ou en incluant d'autres playbooks ou tâches à partir de fichiers externes, de manière inconditionnelle ou sur la base d'un test conditionnel.

## **Chapitre 8, Simplification des playbooks avec des rôles**

Utiliser les rôles Ansible pour développer plus rapidement des playbooks et réutiliser le code Ansible.

- Décrire ce qu'est un rôle, comment il est structuré et comment vous pouvez l'utiliser dans un playbook.
- Créer un rôle dans le répertoire de projet d'un playbook et l'exécuter dans le cadre de l'un des plays du playbook.
- Sélectionner et récupérer des rôles à partir d'Ansible Galaxy ou d'autres sources, telles qu'un référentiel Git, et les utiliser dans vos playbooks.
- Écrire des playbooks qui tirent parti des rôles système Red Hat Enterprise Linux pour effectuer des opérations standard.

## **Chapitre 9, Résolution des problèmes liés à Ansible**

Résoudre les problèmes liés aux playbooks et hôtes gérés.

- Résoudre les problèmes génériques avec un nouveau playbook et les réparer.
- Résoudre les échecs sur les hôtes gérés lors de l'exécution d'un playbook.

## **Chapitre 10, Automatisation des tâches d'administration Linux**

Automatiser les tâches d'administration courantes de systèmes Linux avec Ansible.

- S'abonner aux systèmes, configurer les canaux de logiciels et les référentiels, et gérer les paquetages RPM sur des hôtes gérés.
- Gérer les utilisateurs et les groupes Linux, configurer SSH et modifier la configuration de Sudo sur les hôtes gérés.
- Gérer le démarrage du service, planifier les processus avec at, cron et systemd, redémarrer et contrôler la cible de démarrage par défaut sur les hôtes gérés.
- Partitionner les périphériques de stockage, configurer LVM, formater les partitions ou les volumes logiques, monter les systèmes de fichiers et ajouter des fichiers ou des espaces d'échange.

## ► OPEN LAB

# DÉPLOIEMENT D'ANSIBLE

Dans cette révision, vous allez installer Ansible sur **workstation** puis l'utiliser comme nœud de contrôle et le configurer pour des connexions aux hôtes gérés **servera** et **serverb**. Utilisez des commandes ad hoc pour réaliser des actions sur des hôtes gérés.

## RÉSULTATS

Vous devez pouvoir :

- Installer Ansible.
- Utiliser des commandes ad hoc pour réaliser des actions sur des hôtes gérés.

Connectez-vous en tant qu'utilisateur **student** sur **workstation**, puis exécutez **lab review-deploy setup**. Ce script vérifie que les hôtes gérés, **servera** et **serverb** sont accessibles sur le réseau. Le script crée une sous-répertoire d'atelier nommé **review-deploy** dans le répertoire personnel du stagiaire.

```
[student@workstation ~]$ lab review-deploy setup
```

## Instructions

Installez et configurez Ansible sur **workstation**. Démontrez que vous pouvez construire les commandes ad hoc spécifiées dans la liste de critères afin de modifier les hôtes gérés et de vérifier que les modifications ont fonctionné comme prévu :

- Installez Ansible sur **workstation** pour qu'il serve de nœud de contrôle.
- Sur le nœud de contrôle, créez un fichier d'inventaire, **/home/student/review-deploy/inventory**, contenant un groupe appelé **dev**. Ce groupe doit être composé des hôtes gérés **servera.lab.example.com** et **serverb.lab.example.com**.
- Créez le fichier de configuration Ansible dans **/home/student/review-deploy/ansible.cfg**. Le fichier de configuration doit pointer vers le fichier d'inventaire **/home/student/review-deploy/inventory**.
- Exécutez une commande ad hoc en utilisant l'augmentation de privilèges pour modifier les contenus du fichier **/etc/motd** sur **servera** et **serverb**, de sorte qu'il contienne la chaîne « **Managed by Ansible\n** » (Géré par Ansible). Utilisez **devops** comme utilisateur distant.
- Exécutez une commande ad hoc pour vérifier que les contenus du fichier **/etc/motd** sur **servera** et **serverb** sont identiques.

## Évaluation

À partir du poste de travail (**workstation**), exécutez le script **lab review-deploy** avec l'argument **grade** pour confirmer que l'exercice a été réussi. Corrigez toute erreur signalée et répétez le script tant que des erreurs persistent.

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

## ► SOLUTION

# DÉPLOIEMENT D'ANSIBLE

Dans cette révision, vous allez installer Ansible sur **workstation** puis l'utiliser comme nœud de contrôle et le configurer pour des connexions aux hôtes gérés **servera** et **serverb**. Utilisez des commandes ad hoc pour réaliser des actions sur des hôtes gérés.

## RÉSULTATS

Vous devez pouvoir :

- Installer Ansible.
- Utiliser des commandes ad hoc pour réaliser des actions sur des hôtes gérés.

Connectez-vous en tant qu'utilisateur **student** sur **workstation**, puis exécutez **lab review-deploy setup**. Ce script vérifie que les hôtes gérés, **servera** et **serverb** sont accessibles sur le réseau. Le script crée une sous-répertoire d'atelier nommé **review-deploy** dans le répertoire personnel du stagiaire.

```
[student@workstation ~]$ lab review-deploy setup
```

## Instructions

Installez et configurez Ansible sur **workstation**. Démontrez que vous pouvez construire les commandes ad hoc spécifiées dans la liste de critères afin de modifier les hôtes gérés et de vérifier que les modifications ont fonctionné comme prévu :

- Installez Ansible sur **workstation** pour qu'il serve de nœud de contrôle.
  - Sur le nœud de contrôle, créez un fichier d'inventaire, **/home/student/review-deploy/inventory**, contenant un groupe appelé **dev**. Ce groupe doit être composé des hôtes gérés **servera.lab.example.com** et **serverb.lab.example.com**.
  - Créez le fichier de configuration Ansible dans **/home/student/review-deploy/ansible.cfg**. Le fichier de configuration doit pointer vers le fichier d'inventaire **/home/student/review-deploy/inventory**.
  - Exécutez une commande ad hoc en utilisant l'augmentation de privilèges pour modifier les contenus du fichier **/etc/motd** sur **servera** et **serverb**, de sorte qu'il contienne la chaîne « **Managed by Ansible\n** » (Géré par Ansible). Utilisez **devops** comme utilisateur distant.
  - Exécutez une commande ad hoc pour vérifier que les contenus du fichier **/etc/motd** sur **servera** et **serverb** sont identiques.
1. Installez Ansible sur **workstation** pour qu'il puisse servir le nœud de contrôle.

```
[student@workstation ~]$ sudo yum install ansible
[sudo] password for student:
Loaded plugins: langpacks, search-disabled-repos
```

```

Resolving Dependencies
--> Running transaction check
--> Package ansible.noarch 0:2.7.1-1.el7ae will be installed
--> Processing Dependency: sshpass for package: ansible-2.7.1-1.el7ae.noarch
...output omitted...
Dependencies Resolved

=====
Package           Arch    Version      Repository  Size
=====
Installing:
ansible          noarch  2.7.1-1.el7ae  ansible      11 M
Installing for dependencies:
...output omitted...
Is this ok [y/d/N]: y
Downloading packages:
...output omitted...

Installed:
ansible.noarch 0:2.7.1-1.el7ae
...output omitted...

```

2. Sur le nœud de contrôle, créez un fichier d'inventaire, **/home/student/review-deploy/inventory**, contenant un groupe appelé dev. Ce groupe doit être composé des hôtes gérés servera.lab.example.com et serverb.lab.example.com.

- 2.1. Utilisez l'éditeur de texte Vim pour créer et modifier le fichier d'inventaire **/home/student/review-deploy/inventory**.

```
[student@workstation ~]$ cd /home/student/review-deploy
[student@workstation review-deploy]$ vim inventory
```

- 2.2. Ajoutez les entrées suivantes au fichier pour créer le groupe dev et les membres servera.lab.example.com et serverb.lab.example.com.  
Enregistrez les modifications et quittez l'éditeur de texte.

```
[dev]
servera.lab.example.com
serverb.lab.example.com
```

3. Créez le fichier de configuration Ansible dans **/home/student/review-deploy/ansible.cfg**. Le fichier de configuration doit pointer vers le fichier d'inventaire **/home/student/review-deploy/inventory**.
- 3.1. Utilisez l'éditeur de texte Vim pour créer et modifier le fichier de configuration Ansible **/home/student/review-deploy/ansible.cfg**.

```
[student@workstation review-deploy]$ vim ansible.cfg
```

- 3.2. Ajoutez les entrées suivantes pour configurer le fichier d'inventaire **./inventory** comme source d'inventaire. Enregistrez les modifications et quittez l'éditeur de texte.

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

4. Exécutez une commande ad hoc en utilisant l'augmentation de priviléges pour modifier les contenus du fichier **/etc/motd** sur **servera** et **serverb**, de sorte qu'il contienne la chaîne « **Managed by Ansible\n** » (Géré par Ansible). Utilisez **devops** comme utilisateur distant.
- 4.1. Depuis le répertoire de projet **/home/student/review-deploy**, exécutez une commande ad hoc en utilisant l'augmentation de priviléges pour modifier les contenus du fichier **/etc/motd** sur **servera** et **serverb**, de sorte qu'il contienne la chaîne « **Managed by Ansible\n** » (Géré par Ansible). Utilisez **devops** comme utilisateur distant.

```
[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 => {
    "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
}
serverb.lab.example.com | CHANGED => {
    "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  
}
```

5. Exécutez une commande ad hoc pour vérifier que les contenus du fichier **/etc/motd** sur **servera** et **serverb** sont identiques.

```
[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
```

## Évaluation

À partir du poste de travail (**workstation**), exécutez le script **lab review-deploy** avec l'argument **grade** pour confirmer que l'exercice a été réussi. Corrigez toute erreur signalée et répétez le script tant que des erreurs persistent.

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

## ► OPEN LAB

# CRÉATION DE PLAYBOOKS

Dans cette révision, vous allez créer trois playbooks dans le répertoire du projet Ansible, **/home/student/review-playbooks**. Un playbook va s'assurer que *lftp* est installé sur des systèmes qui devraient être des clients FTP, un playbook va s'assurer que *vsftpd* est installé et configuré sur des systèmes devant être des serveurs FTP, et un playbook (**site.yml**) va exécuter les deux autres playbooks.

## RÉSULTATS

Vous devez pouvoir :

- Créer et exécuter des playbooks pour effectuer des tâches sur des hôtes gérés.
- Utiliser des modèles, des variables et des gestionnaires Jinja2 dans les playbooks.

Configurez vos ordinateurs pour effectuer cet exercice en vous connectant à **workstation** en tant que **student** et exécutez la commande suivante :

```
[student@workstation ~]$ lab review-playbooks setup
```

## Instructions

Créez un inventaire statique dans **review-playbooks/inventory** avec **serverc.lab.example.com** dans le groupe **ftpclients**, et **serverb.lab.example.com** et **serverd.lab.example.com** dans le groupe **ftpservers**. Créez un fichier **review-playbooks/ansible.cfg** qui configure votre projet Ansible pour utiliser cet inventaire. Vous trouverez peut-être utile de consulter le fichier **/etc/ansible/ansible.cfg** du système pour obtenir de l'aide sur la syntaxe.

Configurez votre projet Ansible pour qu'il se connecte aux hôtes de l'inventaire en utilisant l'utilisateur distant, **devops**, et la méthode **sudo** pour l'augmentation des priviléges. Vous disposez de clés SSH pour vous connecter en tant que **devops** déjà configuré. L'utilisateur **devops** n'a pas besoin de mot de passe pour l'augmentation de priviléges avec **sudo**.

Créez un playbook nommé **ftpclients.yml** dans le répertoire **review-playbooks** qui contient un hôte de ciblage de play dans le groupe d'inventaire **ftpclients**. Vérifiez que le paquetage *lftp* est installé.

Créez un second playbook nommé **ansible-vsftpd.yml** dans le répertoire **review-playbooks** qui contient un hôte de ciblage de play dans le groupe d'inventaire **ftpservers**. Il doit être écrit comme suit :

- Vous avez un fichier de configuration pour **vsftpd** généré à partir d'un modèle Jinja2. Créez un répertoire pour les modèles, **review-playbooks/templates**, et copiez le fichier **vsftpd.conf.j2** fourni dans celui-ci. Créez également le répertoire **review-playbooks/vars**. Copiez dans ce répertoire le fichier **defaults-template.yml** fourni, qui contient les paramètres de variable par défaut utilisés pour compléter ce modèle lors de son déploiement.

- Créez un fichier de variable, **review-playbooks/vars/vars.yml**, qui définit trois variables :

| VARIABLE           | VALEUR                  |
|--------------------|-------------------------|
| vsftpd_package     | vsftpd                  |
| vsftpd_service     | vsftpd                  |
| vsftpd_config_file | /etc/vsftpd/vsftpd.conf |

- Dans votre playbook **ansible-vsftpd.yml**, assurez-vous que vous utilisez **vars\_files** pour inclure les fichiers de variables dans le répertoire **review-playbooks/vars** de votre play.
- Dans le play dans **ansible-vsftpd.yml**, créez des tâches qui :
  - Vérifiez que le paquetage listé par la variable `vsftpd_package` est installé.
  - Vérifiez que le service listé par la variable `vsftpd_service` est démarré et activé pour démarrer au démarrage.
  - Utilisez le module `template` pour déployer le modèle **templates/vsftpd.conf.j2** à l'emplacement défini par la variable `vsftpd_config_file`. Le fichier doit appartenir à l'utilisateur `root`, au groupe `root`, avoir des autorisations de fichier octal 0600 et avoir le type SELinux `etc_t`. Notifiez un gestionnaire qui redémarre `vsftpd` si cette tâche provoque une modification.
  - Assurez-vous que le paquetage `firewalld` est installé et que le service est démarré et activé. Assurez-vous que `firewalld` a été configuré pour autoriser immédiatement et définitivement les connexions au service FTP.
- Dans votre playbook **ansible-vsftpd.yml**, créez un gestionnaire pour redémarrer les services listés par la variable `vsftpd_service` lors de la notification.

Créez un troisième playbook, **site.yml**, dans le répertoire **review-playbooks**. Ce playbook ne doit importer que les deux autres cahiers.

Nous vous encourageons à suivre les pratiques de playbook recommandées en nommant tous vos plays et tâches. Les playbooks doivent être écrits en utilisant des modules appropriés, et doivent pouvoir être réexécutés en toute sécurité. Les playbooks ne doivent pas apporter de modifications inutiles aux systèmes.

N'oubliez pas d'utiliser la commande **ansible-doc** pour vous aider à trouver des modules et des informations sur la façon de les utiliser.

Lorsque vous avez terminé, vous devez utiliser **ansible-playbook site.yml** pour vérifier votre travail avant d'exécuter le script de notation. Vous pouvez également exécuter les playbooks séparément pour vous assurer qu'ils fonctionnent.



### IMPORTANT

Si vous rencontrez des problèmes avec votre playbook **site.yml**, assurez-vous que **ansible-vsftpd.yml** et **ftpclients.yml** présentent une mise en retrait cohérente les uns avec les autres.

## Évaluation

En tant qu'utilisateur student sur workstation, exécutez la commande **lab review-playbooks grade** pour confirmer que l'exercice est réussi. Corrigez toute erreur signalée et répétez le script tant que des erreurs persistent.

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

## Nettoyage

Exécutez la commande **lab review-playbooks cleanup** pour effacer les tâches de l'atelier sur serverb, serverc et serverd.

```
[student@workstation ~]$ lab review-playbooks cleanup
```

## ► SOLUTION

# CRÉATION DE PLAYBOOKS

Dans cette révision, vous allez créer trois playbooks dans le répertoire du projet Ansible, `/home/student/review-playbooks`. Un playbook va s'assurer que `lftp` est installé sur des systèmes qui devraient être des clients FTP, un playbook va s'assurer que `vsftpd` est installé et configuré sur des systèmes devant être des serveurs FTP, et un playbook (`site.yml`) va exécuter les deux autres playbooks.

## RÉSULTATS

Vous devez pouvoir :

- Créer et exécuter des playbooks pour effectuer des tâches sur des hôtes gérés.
- Utiliser des modèles, des variables et des gestionnaires Jinja2 dans les playbooks.

Configurez vos ordinateurs pour effectuer cet exercice en vous connectant à `workstation` en tant que `student` et exécutez la commande suivante :

```
[student@workstation ~]$ lab review-playbooks setup
```

## Instructions

Créez un inventaire statique dans `review-playbooks/inventory` avec `serverc.lab.example.com` dans le groupe `ftpclients`, et `serverb.lab.example.com` et `serverd.lab.example.com` dans le groupe `ftpservers`. Créez un fichier `review-playbooks/ansible.cfg` qui configure votre projet Ansible pour utiliser cet inventaire. Vous trouverez peut-être utile de consulter le fichier `/etc/ansible/ansible.cfg` du système pour obtenir de l'aide sur la syntaxe.

Configurez votre projet Ansible pour qu'il se connecte aux hôtes de l'inventaire en utilisant l'utilisateur distant, `devops`, et la méthode `sudo` pour l'augmentation des priviléges. Vous disposez de clés SSH pour vous connecter en tant que `devops` déjà configuré. L'utilisateur `devops` n'a pas besoin de mot de passe pour l'augmentation de priviléges avec `sudo`.

Créez un playbook nommé `ftpclients.yml` dans le répertoire `review-playbooks` qui contient un hôte de ciblage de play dans le groupe d'inventaire `ftpclients`. Vérifiez que le paquetage `lftp` est installé.

Créez un second playbook nommé `ansible-vsftpd.yml` dans le répertoire `review-playbooks` qui contient un hôte de ciblage de play dans le groupe d'inventaire `ftpservers`. Il doit être écrit comme suit :

- Vous avez un fichier de configuration pour `vsftpd` généré à partir d'un modèle Jinja2. Créez un répertoire pour les modèles, `review-playbooks/templates`, et copiez le fichier `vsftpd.conf.j2` fourni dans celui-ci. Créez également le répertoire `review-playbooks/vars`. Copiez dans ce répertoire le fichier `defaults-template.yml` fourni, qui contient les paramètres de variable par défaut utilisés pour compléter ce modèle lors de son déploiement.

- Créez un fichier de variable, **review-playbooks/vars/vars.yml**, qui définit trois variables :

| VARIABLE           | VALEUR                  |
|--------------------|-------------------------|
| vsftpd_package     | vsftpd                  |
| vsftpd_service     | vsftpd                  |
| vsftpd_config_file | /etc/vsftpd/vsftpd.conf |

- Dans votre playbook **ansible-vsftpd.yml**, assurez-vous que vous utilisez **vars\_files** pour inclure les fichiers de variables dans le répertoire **review-playbooks/vars** de votre play.
- Dans le play dans **ansible-vsftpd.yml**, créez des tâches qui :
  1. Vérifiez que le paquetage listé par la variable `{} vsftpd_package {}` est installé.
  2. Vérifiez que le service listé par la variable `{} vsftpd_service {}` est démarré et activé pour démarrer au démarrage.
  3. Utilisez le module **template** pour déployer le modèle **templates/vsftpd.conf.j2** à l'emplacement défini par la variable `{} vsftpd_config_file {}`. Le fichier doit appartenir à l'utilisateur **root**, au groupe **root**, avoir des autorisations de fichier octal **0600** et avoir le type SELinux **etc\_t**. Notifyez un gestionnaire qui redémarre **vsftpd** si cette tâche provoque une modification.
  4. Assurez-vous que le paquetage **firewalld** est installé et que le service est démarré et activé. Assurez-vous que **firewalld** a été configuré pour autoriser immédiatement et définitivement les connexions au service FTP.
- Dans votre playbook **ansible-vsftpd.yml**, créez un gestionnaire pour redémarrer les services listés par la variable `{} vsftpd_service {}` lors de la notification.

Créez un troisième playbook, **site.yml**, dans le répertoire **review-playbooks**. Ce playbook ne doit importer que les deux autres cahiers.

Nous vous encourageons à suivre les pratiques de playbook recommandées en nommant tous vos plays et tâches. Les playbooks doivent être écrits en utilisant des modules appropriés, et doivent pouvoir être réexécutés en toute sécurité. Les playbooks ne doivent pas apporter de modifications inutiles aux systèmes.

N'oubliez pas d'utiliser la commande **ansible-doc** pour vous aider à trouver des modules et des informations sur la façon de les utiliser.

Lorsque vous avez terminé, vous devez utiliser **ansible-playbook site.yml** pour vérifier votre travail avant d'exécuter le script de notation. Vous pouvez également exécuter les playbooks séparément pour vous assurer qu'ils fonctionnent.



#### IMPORTANT

Si vous rencontrez des problèmes avec votre playbook **site.yml**, assurez-vous que **ansible-vsftpd.yml** et **ftpclients.yml** présentent une mise en retrait cohérente les uns avec les autres.

1. En tant qu'utilisateur **student** sur **workstation**, créez le fichier d'inventaire **/home/student/review-playbooks/inventory**, contenant **serverc.lab.example.com**

dans le groupe **ftpclients**, et `serverb.lab.example.com` et `serverd.lab.example.com` dans le groupe **ftpservers**.

- 1.1. Changez le répertoire dans le répertoire de projet Ansible, `/home/student/review-playbooks`, créé par le script d'installation.

```
[student@workstation ~]$ cd /home/student/review-playbooks
```

- 1.2. Créez le fichier d'inventaire statique, **inventory**, en l'ouvrant avec un éditeur de texte.

```
[student@workstation review-playbooks]$ vim inventory
```

- 1.3. Remplissez le fichier **inventory** avec le contenu suivant :

```
[ftpservers]
serverb.lab.example.com
serverd.lab.example.com
```

```
[ftpclients]
serverc.lab.example.com
```

- 1.4. Enregistrez les modifications dans le fichier d'inventaire nouvellement créé.

2. Créez le fichier de configuration Ansible, `/home/student/review-playbooks/ansible.cfg`, et remplissez-le avec les entrées nécessaires pour répondre à ces exigences :

- Configurer le projet Ansible pour utiliser l'inventaire nouvellement créé
- Se connecter aux hôtes gérés en tant qu'utilisateur `devops`
- Utiliser l'augmentation de priviléges en utilisant `sudo` comme utilisateur `root`
- Augmenter les priviléges pour chaque tâche par défaut

- 2.1. Créez le fichier de configuration Ansible, `/home/student/review-playbooks/ansible.cfg`, en l'ouvrant avec un éditeur de texte.

```
[student@workstation review-playbooks]$ vim ansible.cfg
```

- 2.2. Configurez l'inventaire, l'utilisateur distant, la méthode d'augmentation des priviléges et l'utilisateur pour le projet Ansible en ajoutant les entrées suivantes dans le fichier de configuration `ansible.cfg`.

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

[privilegeEscalation]
become_user = root
become_method = sudo
```

```
become = true
```

- 2.3. Enregistrez les modifications dans le fichier de configuration Ansible nouvellement créé.
3. Créez le playbook, **/home/student/review-playbooks/ftpclients.yml**, contenant un play ciblant les hôtes du groupe d'inventaire **ftpclients** et vérifiez que **lftp** est installé.
  - 3.1. Créez le fichier de playbook, **/home/student/review-playbooks/ftpclients.yml**, en l'ouvrant avec un éditeur de texte.

```
[student@workstation review-playbooks]$ vim ftpclients.yml
```

- 3.2. Remplissez le nouveau fichier de playbook avec un play pour vous assurer que le paquetage **lftp** est installé sur les hôtes du groupe d'inventaire **ftpclients** en ajoutant les entrées suivantes.

```
---
- name: Ensure FTP Client Configuration
  hosts: ftpclients

  tasks:
    - name: latest version of lftp is installed
      yum:
        name: lftp
        state: latest
```

- 3.3. Enregistrez les modifications dans le fichier de playbook nouvellement créé.
4. Placez le fichier de configuration **vsftpd** fourni dans le sous-répertoire **templates**.
  - 4.1. Créez le sous-répertoire **templates**.

```
[student@workstation review-playbooks]$ mkdir -v templates
mkdir: created directory 'templates'
```

- 4.2. Déplacez le fichier **vsftpd.conf.j2** dans le sous-répertoire **templates** qui vient d'être créé.
5. Placez le fichier **defaults-template.yml** fourni dans le sous-répertoire **vars**.
  - 5.1. Créez le sous-répertoire **vars**.

```
[student@workstation review-playbooks]$ mv -v vsftpd.conf.j2 templates
'vsftpd.conf.j2' -> 'templates/vsftpd.conf.j2'
```

- 5.5. Placez le fichier **vsftpd.conf.j2** dans le sous-répertoire **templates**.

- 5.6. Créez le sous-répertoire **vars**.

```
[student@workstation review-playbooks]$ mkdir -v vars
```

```
mkdir: created directory 'vars'
```

- 5.2. Déplacez le fichier **defaults-template.yml** dans le sous-répertoire **vars** qui vient d'être créé.

```
[student@workstation review-playbooks]$ mv -v defaults-template.yml vars  
'defaults-template.yml' -> 'vars/defaults-template.yml'
```

6. Créez un fichier de définition de variable **vars.yml** dans le sous-répertoire **vars** pour définir les trois variables suivantes et leurs valeurs.

| VARIABLE           | VALEUR                  |
|--------------------|-------------------------|
| vsftpd_package     | vsftpd                  |
| vsftpd_service     | vsftpd                  |
| vsftpd_config_file | /etc/vsftpd/vsftpd.conf |

- 6.1. Créez le fichier **/home/student/review-playbooks/vars/vars.yml**.

```
[student@workstation review-playbooks]$ vim vars/vars.yml
```

- 6.2. Remplissez le fichier **vars.yml** avec les définitions de variables suivantes.

```
vsftpd_package: vsftpd  
vsftpd_service: vsftpd  
vsftpd_config_file: /etc/vsftpd/vsftpd.conf
```

- 6.3. Enregistrez les modifications dans le fichier de définition de variable nouvellement créé.

7. En utilisant le modèle Jinja2 précédemment créé et les fichiers de définition de variable, créez un deuxième playbook, **/home/student/review-playbooks/ansible-vsftpd.yml**, pour configurer le service vsftpd sur les hôtes du groupe d'inventaire **ftpservers**.

- 7.1. Créez le fichier de playbook, **/home/student/review-playbooks/ansible-vsftpd.yml**, en l'ouvrant avec un éditeur de texte.

```
[student@workstation review-playbooks]$ vim ansible-vsftpd.yml
```

- 7.2. Remplissez le nouveau fichier de playbook avec les entrées suivantes afin de configurer le service vsftpd sur les hôtes du groupe d'inventaire **ftpservers**.

```
---  
- name: FTP server is installed  
hosts:  
  - ftpservers  
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

handlers:
  - name: restart vsftpd
    service:
      name: "{{ vsftpd_service }}"
      state: restarted

```

7.3. Enregistrez les modifications dans le fichier de playbook nouvellement créé.

8. Créez un troisième playbook, **/home/student/review-playbooks/site.yml**, et incluez les plays des deux playbooks créés précédemment, **ftpclients.yml** et **ansible-vsftpd.yml**.

- 8.1. Créez le fichier de playbook, **/home/student/review-playbooks/site.yml**, en l'ouvrant avec un éditeur de texte.

```
[student@workstation review-playbooks]$ vim site.yml
```

- 8.2. Remplissez le nouveau fichier de playbook avec les entrées suivantes afin d'inclure les plays des deux autres playbooks.

```
# FTP Servers playbook
- import_playbook: ansible-vsftpd.yml

# FTP Clients playbook
- import_playbook: ftpclients.yml
```

- 8.3. Enregistrez les modifications dans le fichier de playbook nouvellement créé.

9. Exécutez le playbook **/home/student/review-playbooks/site.yml** pour vérifier qu'il exécute les tâches souhaitées sur les hôtes gérés.

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

## Évaluation

En tant qu'utilisateur **student** sur **workstation**, exécutez la commande **lab review-playbooks grade** pour confirmer que l'exercice est réussi. Corrigez toute erreur signalée et répétez le script tant que des erreurs persistent.

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

## Nettoyage

Exécutez la commande **lab review-playbooks cleanup** pour effacer les tâches de l'atelier sur **serverb**, **serverc** et **serverd**.

```
[student@workstation ~]$ lab review-playbooks cleanup
```

## ► OPEN LAB

# CRÉATION DE RÔLES ET UTILISATION DE L'INVENTAIRE DYNAMIQUE

Dans cette révision, vous allez convertir le playbook **ansible-vsftpd.yml** en rôle, puis utiliser ce rôle dans un nouveau playbook qui exécutera également des tâches supplémentaires. Vous allez également installer et utiliser un script d'inventaire dynamique qui vous sera fourni.

## RÉSULTATS

Vous devez pouvoir réaliser les tâches suivantes :

- Créer un rôle pour configurer le service vsftpd en utilisant les tâches d'un playbook existant
- Inclure un rôle dans un playbook et l'exécuter
- Utiliser un inventaire dynamique pour exécuter un playbook

Connectez-vous à **workstation** en tant qu'utilisateur **student** avec le mot de passe **student**.

Sur **workstation**, exécutez le script **lab review-roles setup**. Ce script permet de s'assurer que les hôtes distants sont accessibles sur le réseau. Le script vérifie également qu'Ansible est installé sur **workstation**, crée une structure de répertoire pour l'environnement de l'atelier et installe les fichiers d'atelier requis.

```
[student@workstation ~]$ lab review-roles setup
```

## Instructions

Configurez votre projet Ansible pour qu'il utilise le script d'inventaire dynamique **crinventory.py** et le fichier d'inventaire statique **inventory**.

Convertissez le playbook **ansible-vsftpd.yml** dans le rôle **ansible-vsftpd**, comme spécifié ci-dessous :

- Utilisez la commande **ansible-galaxy** pour créer la structure de répertoire pour le rôle **ansible-vsftpd** dans le répertoire **review-roles/roles** de votre projet Ansible.
- Le fichier **defaults-template.yml** contient des variables par défaut pour le rôle. Il doit être déplacé vers un emplacement approprié dans la structure du répertoire de rôles.
- Le fichier **vars.yml** contient des variables régulières pour le rôle. Il doit être déplacé vers un emplacement approprié dans la structure du répertoire de rôles.
- Le modèle **vsftpd.conf.j2** doit être déplacé vers un emplacement approprié dans la structure du répertoire de rôles.
- Les tâches et les gestionnaires du playbook **ansible-vsftpd.yml** doivent être correctement installés dans le rôle.

- Vous pouvez modifier le fichier **meta/main.yml** du rôle pour définir les champs d'auteur, de description et de licence (utilisez BSD pour la licence). Vous pouvez également modifier le fichier **README.md** comme vous le souhaitez à des fins d'exhaustivité.
- Supprimez tout sous-répertoire dans le rôle que vous n'utilisez pas.

Créez un nouveau playbook, **vsftpd-configure.yml**, dans le répertoire **review-roles**. Il doit être écrit comme suit :

- Il doit contenir un jeu ciblant les hôtes dans le groupe d'inventaire **ftpservers**.
- Le play doit définir les variables suivantes :

| VARIABLE          | VALEUR     |
|-------------------|------------|
| vsftpd_anon_root  | /mnt/share |
| vsftpd_local_root | /mnt/share |

- Le play doit appliquer le rôle **ansible-vsftpd**.
- Le play doit inclure les tâches suivantes dans l'ordre spécifié :
  - Utilisez le module **command** pour une étiquette de disque GPT sur **/dev/vdb**, qui démarre 1 MiB après le début du périphérique et s'arrête à la fin du périphérique. Utilisez la commande **ansible-doc** afin d'apprendre à utiliser l'argument **creates** pour ignorer cette tâche si **/dev/vdb1** a déjà été créé. Cela évite tout repartitionnement destructif du périphérique. Utilisez la commande suivante pour créer la partition : **parted --script /dev/vdb mklabel gpt mkpart primary 1MiB 100%**
  - Assurez-vous qu'un répertoire **/mnt/share** existe pour être utilisé en tant que point de montage.
  - Utilisez **ansible-doc -l** pour trouver un module qui peut créer un système de fichiers sur un périphérique par blocs. Utilisez **ansible-doc** pour apprendre à utiliser ce module. Ajoutez une tâche au playbook qui utilise un système de fichiers XFS sur **/dev/vdb1**. Ne forcez pas la création de ce système de fichiers s'il en existe déjà un.
  - Ajoutez une tâche qui s'assure que **/etc/fstab** monte le périphérique **/dev/vdb1** sur **/mnt/share** au démarrage, et qu'il est actuellement monté. (Utilisez **ansible-doc** pour trouver un module qui peut vous aider.) Si cette tâche est modifiée,通知ez le gestionnaire du rôle **ansible-vsftpd** qui redémarre vsftpd.
  - Ajoutez une tâche qui garantit que le répertoire **/mnt/share** est la propriété de l'utilisateur **root** et du groupe **root**, a le type SELinux défini dans la variable **{{ vsftpd\_setype }}** du rôle et possède les autorisations octales 0755. Cela doit être fait après que le système de fichiers est monté pour définir les autorisations sur le système de fichiers monté, et non sur le répertoire de point de montage d'espace réservé.
  - Vérifiez qu'un fichier nommé **README** existe dans le répertoire spécifié par **{{ vsftpd\_anon\_root }}** et qu'il contient la chaîne **Welcome to the FTP server at serverX.lab.example.com** à l'endroit où **serverX.lab.example.com** correspond au nom d'hôte complet de ce serveur. Ce fichier doit avoir les permissions octales 0644 et le type SELinux spécifié par la variable **{{ vsftpd\_setype }}**. (Conseil : regardez les modules **copy** ou **template** et les faits Ansible disponibles pour résoudre ce problème.)

**IMPORTANT**

Vous pouvez trouver utile de déboguer votre rôle en le testant dans un playbook qui ne contient pas les tâches supplémentaires ou les variables de playbook listées ci-dessus, mais qui contient uniquement un play ciblant des hôtes dans le groupe `ftpservers` et applique le rôle.

Une fois que vous avez confirmé qu'un playbook simplifié utilisant uniquement le rôle fonctionne exactement comme le playbook `ansible-vsftpd.yml`d'origine, vous pouvez créer le playbook `vsftpd-configure.yml` complet en ajoutant les variables et tâches supplémentaires spécifiées ci-dessus.

Modifiez le playbook `review-roles/site.yml` pour utiliser le nouveau playbook `vsftpd-configure.yml` au lieu de `ansible-vsftpd.yml`.

Nous vous encourageons à suivre les pratiques de playbook recommandées en nommant tous vos plays et tâches. Les playbooks doivent être écrits en utilisant des modules appropriés, et doivent pouvoir être réexécutés en toute sécurité. Les playbooks ne doivent pas apporter de modifications inutiles aux systèmes.

Lorsque vous avez terminé, utilisez `ansible-playbook site.yml` pour vérifier votre travail avant d'exécuter le script de notation. Vous pouvez également exécuter les playbooks séparément pour vous assurer qu'ils fonctionnent.

**IMPORTANT**

Si vous rencontrez des problèmes avec votre playbook `site.yml`, assurez-vous que `vsftpd-configure.yml` et `ftpclients.yml` présentent une mise en retrait cohérente les uns avec les autres.

## Évaluation

À partir de `workstation`, exécutez la commande `lab review-roles grade` pour confirmer que l'exercice est réussi. Corrigez toute erreur signalée et répétez le script tant que des erreurs persistent.

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

## Nettoyage

Exécutez la commande `lab review-roles cleanup` pour effacer les tâches de l'atelier sur `servera` et `serverb`.

```
[student@workstation ~]$ lab review-roles cleanup
```

## ► SOLUTION

# CRÉATION DE RÔLES ET UTILISATION DE L'INVENTAIRE DYNAMIQUE

Dans cette révision, vous allez convertir le playbook **ansible-vsftpd.yml** en rôle, puis utiliser ce rôle dans un nouveau playbook qui exécutera également des tâches supplémentaires. Vous allez également installer et utiliser un script d'inventaire dynamique qui vous sera fourni.

## RÉSULTATS

Vous devez pouvoir réaliser les tâches suivantes :

- Créer un rôle pour configurer le service vsftpd en utilisant les tâches d'un playbook existant
- Inclure un rôle dans un playbook et l'exécuter
- Utiliser un inventaire dynamique pour exécuter un playbook

Connectez-vous à **workstation** en tant qu'utilisateur **student** avec le mot de passe **student**.

Sur **workstation**, exécutez le script **lab review-roles setup**. Ce script permet de s'assurer que les hôtes distants sont accessibles sur le réseau. Le script vérifie également qu'Ansible est installé sur **workstation**, crée une structure de répertoire pour l'environnement de l'atelier et installe les fichiers d'atelier requis.

```
[student@workstation ~]$ lab review-roles setup
```

## Instructions

Configurez votre projet Ansible pour qu'il utilise le script d'inventaire dynamique **crinventory.py** et le fichier d'inventaire statique **inventory**.

Convertissez le playbook **ansible-vsftpd.yml** dans le rôle **ansible-vsftpd**, comme spécifié ci-dessous :

- Utilisez la commande **ansible-galaxy** pour créer la structure de répertoire pour le rôle **ansible-vsftpd** dans le répertoire **review-roles/roles** de votre projet Ansible.
- Le fichier **defaults-template.yml** contient des variables par défaut pour le rôle. Il doit être déplacé vers un emplacement approprié dans la structure du répertoire de rôles.
- Le fichier **vars.yml** contient des variables régulières pour le rôle. Il doit être déplacé vers un emplacement approprié dans la structure du répertoire de rôles.
- Le modèle **vsftpd.conf.j2** doit être déplacé vers un emplacement approprié dans la structure du répertoire de rôles.
- Les tâches et les gestionnaires du playbook **ansible-vsftpd.yml** doivent être correctement installés dans le rôle.

- Vous pouvez modifier le fichier **meta/main.yml** du rôle pour définir les champs d'auteur, de description et de licence (utilisez BSD pour la licence). Vous pouvez également modifier le fichier **README.md** comme vous le souhaitez à des fins d'exhaustivité.
- Supprimez tout sous-répertoire dans le rôle que vous n'utilisez pas.

Créez un nouveau playbook, **vsftpd-configure.yml**, dans le répertoire **review-roles**. Il doit être écrit comme suit :

- Il doit contenir un jeu ciblant les hôtes dans le groupe d'inventaire `ftpservers`.
- Le play doit définir les variables suivantes :

| VARIABLE                       | VALEUR                  |
|--------------------------------|-------------------------|
| <code>vsftpd_anon_root</code>  | <code>/mnt/share</code> |
| <code>vsftpd_local_root</code> | <code>/mnt/share</code> |

- Le play doit appliquer le rôle `ansible-vsftpd`.
- Le play doit inclure les tâches suivantes dans l'ordre spécifié :
  1. Utilisez le module `command` pour une étiquette de disque GPT sur `/dev/vdb`, qui démarre 1 MiB après le début du périphérique et s'arrête à la fin du périphérique. Utilisez la commande `ansible-doc` afin d'apprendre à utiliser l'argument `creates` pour ignorer cette tâche si `/dev/vdb1` a déjà été créé. Cela évite tout repartitionnement destructif du périphérique. Utilisez la commande suivante pour créer la partition : `parted --script /dev/vdb mklabel gpt mkpart primary 1MiB 100%`
  2. Assurez-vous qu'un répertoire `/mnt/share` existe pour être utilisé en tant que point de montage.
  3. Utilisez `ansible-doc -l` pour trouver un module qui peut créer un système de fichiers sur un périphérique par blocs. Utilisez `ansible-doc` pour apprendre à utiliser ce module. Ajoutez une tâche au playbook qui utilise un système de fichiers XFS sur `/dev/vdb1`. Ne forcez pas la création de ce système de fichiers s'il en existe déjà un.
  4. Ajoutez une tâche qui s'assure que `/etc/fstab` monte le périphérique `/dev/vdb1` sur `/mnt/share` au démarrage, et qu'il est actuellement monté. (Utilisez `ansible-doc` pour trouver un module qui peut vous aider.) Si cette tâche est modifiée,通知ez le gestionnaire du rôle `ansible-vsftpd` qui redémarre vsftpd.
  5. Ajoutez une tâche qui garantit que le répertoire `/mnt/share` est la propriété de l'utilisateur `root` et du groupe `root`, a le type SELinux défini dans la variable `{{ vsftpd_setype }}` du rôle et possède les autorisations octales 0755. Cela doit être fait après que le système de fichiers est monté pour définir les autorisations sur le système de fichiers monté, et non sur le répertoire de point de montage d'espace réservé.
  6. Vérifiez qu'un fichier nommé `README` existe dans le répertoire spécifié par `{{ vsftpd_anon_root }}` et qu'il contient la chaîne `Welcome to the FTP server at serverX.lab.example.com` à l'endroit où `serverX.lab.example.com` correspond au nom d'hôte complet de ce serveur. Ce fichier doit avoir les permissions octales 0644 et le type SELinux spécifié par la variable `{{ vsftpd_setype }}`. (Conseil : regardez les modules `copy` ou `template` et les faits Ansible disponibles pour résoudre ce problème.)

**IMPORTANT**

Vous pouvez trouver utile de déboguer votre rôle en le testant dans un playbook qui ne contient pas les tâches supplémentaires ou les variables de playbook listées ci-dessus, mais qui contient uniquement un play ciblant des hôtes dans le groupe `ftpservers` et applique le rôle.

Une fois que vous avez confirmé qu'un playbook simplifié utilisant uniquement le rôle fonctionne exactement comme le playbook `ansible-vsftpd.yml`d'origine, vous pouvez créer le playbook `vsftpd-configure.yml` complet en ajoutant les variables et tâches supplémentaires spécifiées ci-dessus.

Modifiez le playbook `review-roles/site.yml` pour utiliser le nouveau playbook `vsftpd-configure.yml` au lieu de `ansible-vsftpd.yml`.

Nous vous encourageons à suivre les pratiques de playbook recommandées en nommant tous vos plays et tâches. Les playbooks doivent être écrits en utilisant des modules appropriés, et doivent pouvoir être réexécutés en toute sécurité. Les playbooks ne doivent pas apporter de modifications inutiles aux systèmes.

Lorsque vous avez terminé, utilisez `ansible-playbook site.yml` pour vérifier votre travail avant d'exécuter le script de notation. Vous pouvez également exécuter les playbooks séparément pour vous assurer qu'ils fonctionnent.

**IMPORTANT**

Si vous rencontrez des problèmes avec votre playbook `site.yml`, assurez-vous que `vsftpd-configure.yml` et `ftpclients.yml` présentent une mise en retrait cohérente les uns avec les autres.

1. Connectez-vous à votre hôte `workstation` en tant que `student`. Accédez au répertoire de travail `review-roles`.

```
[student@workstation ~]$ cd ~/review-roles
[student@workstation review-roles]$
```

2. Configurez le projet Ansible pour qu'il utilise à la fois le fichier d'inventaire dynamique `cinventory.py` et le fichier d'inventaire statique `inventory`. Vérifiez la configuration de l'inventaire à l'aide de la commande `ansible-inventory`.

- 2.1. Placez le fichier d'inventaire statique dans un répertoire nommé `inventory`.

```
[student@workstation review-roles]$ mv -v inventory inventory.tmp
'inventory' -> 'inventory.tmp'
[student@workstation review-roles]$ mkdir -v inventory
mkdir: created directory 'inventory'
[student@workstation review-roles]$ mv -v inventory.tmp inventory/inventory
'inventory.tmp' -> 'inventory/inventory'
```

- 2.2. Ajoutez le script d'inventaire dynamique au répertoire d'inventaire. Assurez-vous que le script est exécutable.

```
[student@workstation review-roles]$ chmod 755 -v crinventory.py
mode of 'crinventory.py' changed from 0644 (rw-r--r--) to 0755 (rwxr-xr-x)
[student@workstation review-roles]$ mv -v crinventory.py inventory
'crinventory.py' -> 'inventory/crinventory.py'
```

- 2.3. Configurez le répertoire **inventory** en tant que source des fichiers d'inventaire du projet.

La section **[defaults]** du fichier **ansible.cfg** ressemble à ceci :

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

- 2.4. Utilisez la commande **ansible-inventory --list all** pour vérifier la configuration de l'inventaire du projet :

```
[student@workstation review-roles]$ ansible-inventory --list all
{
    "_meta": {
        "hostvars": {
            "servera.lab.example.com": {},
            "serverb.lab.example.com": {},
            "serverc.lab.example.com": {},
            "serverd.lab.example.com": {}
        }
    },
    "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"
        ]
    },
    "ungrouped": {}
}
```

3. Convertissez le playbook **ansible-vsftpd.yml** en rôle **ansible-vsftpd**.

- 3.1. Créez le sous-répertoire **roles**.

```
[student@workstation review-roles]$ mkdir -v roles
mkdir: created directory 'roles'
```

- 3.2. À l'aide de **ansible-galaxy**, créez la structure de répertoire pour le nouveau rôle **ansible-vsftpd** dans le sous-répertoire **roles**.

```
[student@workstation review-roles]$ cd roles
[student@workstation roles]$ ansible-galaxy init ansible-vsftpd
- ansible-vsftpd was created successfully
[student@workstation roles]$ cd ..
[student@workstation review-roles]$
```

- 3.3. À l'aide de **tree**, vérifiez la structure de répertoire créée pour le nouveau rôle.

```
[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
```

- 3.4. Remplacez le fichier **roles/ansible-vsftpd/defaults/main.yml** par les définitions de variable dans le fichier **defaults-template.yml**.

```
[student@workstation review-roles]$ mv -v defaults-template.yml \
> roles/ansible-vsftpd/defaults/main.yml
'defaults-template.yml' -> 'roles/ansible-vsftpd/defaults/main.yml'
```

- 3.5. Remplacez le fichier **roles/ansible-vsftpd/vars/main.yml** par les définitions de variable dans le fichier **vars.yml**.

```
[student@workstation review-roles]$ mv -v vars.yml \
> roles/ansible-vsftpd/vars/main.yml
```

```
'vars.yml' -> 'roles/ansible-vsftpd/vars/main.yml'
```

- 3.6. Utilisez le fichier **templates/vsftpd.conf.j2** comme modèle pour le rôle **ansible-vsftpd**.

```
[student@workstation review-roles]$ mv -v vsftpd.conf.j2 \
> roles/ansible-vsftpd/templates/
'vsftpd.conf.j2' -> 'roles/ansible-vsftpd/templates/vsftpd.conf.j2'
```

- 3.7. Copiez les tâches dans le playbook **ansible-vsftpd.yml** dans le fichier **roles/ansible-vsftpd/tasks/main.yml**. La valeur du mot-clé **src** dans la tâche du module template n'a plus besoin de référencer le sous-répertoire **templates**. Le fichier **roles/ansible-vsftpd/tasks/main.yml** doit contenir les éléments ci-après une fois que vous avez terminé.

```
---
# 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
```

- 3.8. Copiez les gestionnaires dans le playbook **ansible-vsftpd.yml** dans le fichier **roles/ansible-vsftpd/handlers/main.yml**. Le fichier **roles/ansible-vsftpd/handlers/main.yml** doit contenir les éléments ci-après une fois que vous avez terminé.

```
---
# handlers file for ansible-vsftpd
- name: restart vsftpd
  service:
    name: "{{ vsftpd_service }}"
    state: restarted
```

4. Mettez à jour le contenu du fichier **roles/ansible-vsftpd/meta/main.yml**.

- 4.1. Modifiez la valeur de l'entrée **author** sur **Red Hat Training**.

```
author: Red Hat Training
```

- 4.2. Modifiez la valeur de l'entrée **description** sur « **example role for D0407** ».

```
description: example role for D0407
```

- 4.3. Modifiez la valeur de l'entrée **company** sur « **Red Hat** ».

```
company: Red Hat
```

- 4.4. Modifiez la valeur de l'entrée **license**: sur « **BSD** ».

```
license: BSD
```

5. Modifiez le contenu du fichier **roles/ansible-vsftpd/README.md** afin qu'il fournisse des informations pertinentes sur le rôle. Après modification, le fichier doit avoir le contenu suivant.

```
ansible-vsftpd
=====
Example ansible-vsftpd role from Red Hat's "Automation with Ansible" (D0407)
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)

6. Supprimez les répertoires inutilisés du nouveau rôle.

```
[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/'
```

7. Créez le nouveau playbook **vsftpd-configure.yml**. Il doit contenir les éléments suivants.

```
---
- 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
      command: >
        parted --script /dev/vdb mklabel gpt mkpart primary 1MiB 100%
      args:
        creates: /dev/vdb1

    - name: XFS file system exists on /dev/vdb1
      filesystem:
        dev: /dev/vdb1
        fstype: xfs
        force: no

    - name: anon_root mount point exists
```

```

file:
  path: '{{ vsftpd_anon_root }}'
  state: directory

- name: /dev/vdb1 is mounted on anon_root
  mount:
    name: '{{ 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 }}'

```

8. Modifiez le playbook **site.yml** pour utiliser le playbook **vsftpd-configure.yml** nouvellement créé au lieu du playbook **ansible-vsftpd.yml**. Le fichier doit contenir les éléments ci-après une fois que vous avez terminé.

```

# FTP Servers playbook
- import_playbook: vsftpd-configure.yml

# FTP Clients playbook
- import_playbook: ftpclients.yml

```

9. Vérifiez que le playbook **site.yml** fonctionne comme prévu en l'exécutant avec **ansible-playbook**.

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

## Évaluation

À partir de `workstation`, exécutez la commande **lab review-roles grade** pour confirmer que l'exercice est réussi. Corrigez toute erreur signalée et répétez le script tant que des erreurs persistent.

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

## Nettoyage

Exécutez la commande **lab review-roles cleanup** pour effacer les tâches de l'atelier sur servera et serverb.

```
[student@workstation ~]$ lab review-roles cleanup
```



## ANNEXE A

# SUJETS SUPPLÉMENTAIRES

### PROJET

Enquêter sur des sujets supplémentaires non inclus dans le cours officiel.

# EXAMEN DES OPTIONS DE CONFIGURATION ANSIBLE

---

## OBJECTIFS

Après avoir complété cette section, les stagiaires doivent pouvoir utiliser **ansible-config** pour découvrir et étudier les options de configuration et pour déterminer quelles options ont été modifiées par rapport aux paramètres par défaut.

### Affichage des options de configuration

Si vous voulez savoir quelles options sont disponibles dans le fichier de configuration, utilisez la commande **ansible-config list**. Elle affichera une liste exhaustive des options de configuration disponibles et de leurs paramètres par défaut. Cette liste peut varier en fonction de la version d'Ansible que vous avez installée et du nombre de plug-ins Ansible supplémentaires sur votre nœud de contrôle.

Chaque option affichée par **ansible-config list** aura un certain nombre de paires clé-valeur associées à elle. Ces paires clé-valeur fournissent des informations sur le fonctionnement de cette option. Par exemple, l'option ACTION\_WARNINGS affiche les paires clé-valeur suivantes :

| CLÉ           | VALEUR                                                                                                                                                                                         | OBJET                                                                                                                                                                                                 |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| description   | [Par défaut, Ansible émettra un avertissement à la réception d'une action de tâche (module ou plug-in d'action). Ces avertissements peuvent être supprimés en réglant ce paramètre sur False.] | Décrit l'usage de cette option de configuration.                                                                                                                                                      |
| type          | <b>boolean</b>                                                                                                                                                                                 | Type pour l'option : <b>boolean</b> correspond à une valeur vrai-faux.                                                                                                                                |
| default       | <b>true</b>                                                                                                                                                                                    | La valeur par défaut pour cette option.                                                                                                                                                               |
| version_added | <b>2.5</b>                                                                                                                                                                                     | La version d'Ansible qui a ajouté cette option, à des fins de compatibilité ascendante.                                                                                                               |
| ini           | { <b>clé</b> : <b>action_warnings</b> , <b>section</b> : <b>defaults</b> }                                                                                                                     | Indique la section du fichier d'inventaire de type INI qui contient cette option et le nom de l'option dans le fichier de configuration ( <b>action_warnings</b> , dans la section <b>defaults</b> ). |

| CLÉ | VALEUR                  | OBJET                                                                                                                              |
|-----|-------------------------|------------------------------------------------------------------------------------------------------------------------------------|
| env | ANSIBLE_ACTION_WARNINGS | Si cette variable d'environnement est définie, elle remplacera tout paramètre de l'option défini dans le fichier de configuration. |

## Détermination des options de configuration modifiées

Lorsque vous travaillez avec des fichiers de configuration, vous voudrez peut-être savoir quelles options ont été définies sur des valeurs différentes de celles par défaut.

Vous pouvez le faire en lançant la commande **ansible-config dump -v -\only-changed**. L'option **-v** affiche l'emplacement du fichier **ansible.cfg** utilisé lors du traitement de la commande. La commande **ansible-config** suit le même ordre de priorité que celui mentionné précédemment pour la commande **ansible**. La sortie variera en fonction de l'emplacement du fichier **ansible.cfg** et du répertoire à partir duquel la commande **ansible-config** est exécutée.

Dans l'exemple suivant, il existe un seul fichier de configuration ansible situé dans **/etc/ansible/ansible.cfg**. La commande **ansible-config** est d'abord exécutée à partir du répertoire personnel du stagiaire, puis à partir d'un répertoire de travail avec les mêmes résultats :

```
[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']
```

Cependant, si vous avez un fichier personnalisé **ansible.cfg** dans votre répertoire de travail, la même commande affichera les informations en fonction de l'emplacement de leur exécution et du fichier **ansible.cfg** relatif.

```
[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
```



## RÉFÉRENCES

Page de manuel (1)**ansible-config**

### Fichier de configuration : Documentation Ansible

[https://docs.ansible.com/ansible/2.7/installation\\_guide/intro\\_configuration.html](https://docs.ansible.com/ansible/2.7/installation_guide/intro_configuration.html)

## ANNEXE B

# LICENCES ANSIBLE LIGHTBULB

# LICENCE ANSIBLE LIGHTBULB

---

Certaines parties de ce cours sont des adaptations du projet Ansible Lightbulb. Le support d'origine de ce projet est disponible sur <https://github.com/ansible/lightbulb> sous la licence MIT suivante :

Copyright 2017 Red Hat, Inc.

L'avis de copyright ci-dessus et cet avis d'autorisation doivent être inclus dans toutes les copies ou parties substantielles du Logiciel.

Par la présente, la permission est accordée gratuitement à toute personne obtenant une copie de ce logiciel et des fichiers de documentation associés (le « Logiciel »), d'utiliser le Logiciel sans restriction, même sans limitation des droits d'utiliser, de copier, de modifier, de fusionner, de publier, de distribuer, de concéder en sous-licence et/ou de vendre des copies du Logiciel et de permettre aux personnes auxquelles le Logiciel est fourni de le faire, sous réserve des conditions suivantes :

L'avis de copyright ci-dessus et cet avis d'autorisation doivent être inclus dans toutes les copies ou parties substantielles du Logiciel.

LE LOGICIEL EST FOURNI « EN L'ÉTAT », SANS GARANTIE D'AUCUNE SORTE, EXPRESSE OU IMPLICITE, Y COMPRIS, MAIS SANS S'Y LIMITER, LES GARANTIES DE QUALITÉ MARCHANDE, D'ADÉQUATION À UN USAGE PARTICULIER ET DE NON-CONTREFAÇON. EN AUCUN CAS, LES AUTEURS OU LES DÉTENTEURS DE COPYRIGHT NE POURRONT ÊTRE TENUS RESPONSABLES D'UNE RÉCLAMATION, DE DOMMAGES OU D'AUTRES RESPONSABILITÉS, QUE CE SOIT DANS LE CADRE D'UN CONTRAT, D'UN DÉLIT OU AUTRE, DÉCOULANT DE OU EN RELATION AVEC LE LOGICIEL OU L'UTILISATION OU AUTRES MANIPULATIONS DE CE LOGICIEL.