ANSIBLE

SOLYTI

Sommaire

SOLYTI

Chapitre 1

Introduction

SOLYTI

Les premiers produits répondant aux impératifs de flexibilité de cette nouvelle organisation étaient déjà existants sur le marché. On peut citer notamment des produits comme Puppet, Chef, Salt ou BladeLogic. Malgré leur qualité indéniable, ces produits avaient tout de même quelques défauts :

- Ils demandaient un savoir-faire très spécifique.

- Il fallait absolument installer des agents sur les serveurs à gérer.

- Il n'est pas possible de gérer l'installation initiale de l'agent.

- Enfin, les méthodes d'administration classiques à base de SSH étaient complètement ignorées.

Pour résumer, vous deviez installer des agents supplémentaires, une infrastructure permettant de les gérer, et vous deviez reprendre totalement votre savoir-faire dans un langage très orienté développeur.

Ansible est arrivé un peu après ces pionniers pour répondre justement à ce besoin de remettre les méthodes classiques au goût du jour.

En effet, avec Ansible, vous pourrez vous appuyer sur vos méthodes d'exploitation existantes (SSH, WinRM, Docker, API, etc.) :

- Pas d'agent à installer sur les machines distantes, il faut juste un interpréteur Python.

- Pas d'infrastructure à installer pour gérer les machines, vous pouvez tout faire depuis un poste quelconque avec les accès ad hoc.

Chapitre 2

Installation de Ansible

SOLYTI

Contexte et prérequis

Ce chapitre aborde l'installation d'Ansible et quelques prérequis de sécurité autour de SSH.

Par la suite, vous devrez disposer de quelques prérequis notamment :

- une machine Linux avec un interpréteur Python 2.7 minimum (recommandé Python 3.5) ;

- être administrateur de cette dernière ou avoir des droits suffisants pour lancer des commandes.

SOLYTI

Dans le cas où vous auriez des difficultés à vous fournir en VM, il est possible de passer par d'autres solutions et notamment des conteneurs Docker.

Version de Python

Le support de Python 2 s'est arrêté depuis le 31 décembre 2019. En conséquence, il est fortement recommandé que la machine qui contrôle les déploiements utilise une version Python 3.5 ou supérieure.

Environnement de travail

Par la suite, le support fera mention d'une machine rec-apache-1. Cette dernière représente une machine qui servira pour réaliser les différents tests.

À noter que si vous n'avez pas de DNS, il est tout à fait possible de passer par une adresse IP.

Installation d'Ansible

1. Contexte

L'installation d'Ansible peut se faire de plusieurs manières :

- par l'intermédiaire des packages de votre système ;
- à l'aide de l'outil pip de Python (éventuellement combiné avec virtualenv) ;
- par l'utilisation des archives contenant le code source d'Ansible ;

- ou enfin, en interprétant directement le code source en provenance de Git.

2. Installation derrière un proxy

Si vous êtes derrière un proxy, il faudra exporter les variables http_proxy, https_proxy et ftp_proxy avec la valeur suivante : http://mon-proxy:mon-port. Dans le cas où il faudrait vous identifier, il faudra changer la déclaration de votre proxy pour la valeur suivante : http://identifiant:mdp@mon-proxy:port.

Ces variables sont valables aussi bien pour les utilitaires apt (Debian), yum (Red Hat) ou pip (packaging propre à Python). Ci-dessous les instructions à lancer dans votre terminal pour cette prise en compte :

```
$ export http_proxy=http://proxy:3128
$ export https_proxy=http://proxy:3128
$ export ftp_proxy=http://proxy:3128
```

3. Installation via les packages système

Il existe deux grandes familles de packages dans les distributions Linux : apt pour les dérivés de Debian/Ubuntu et yum (RHEL, CentOS ou Fedora).

a. Installation sous Debian/Ubuntu

Sous Debian, l'installation peut se faire à l'aide d'apt suivi de l'option install et du nom du package. Dans le cas d'Ansible, la commande à lancer sera la suivante :

```
$ sudo apt install ansible
```

b. Installation sur RHEL, CentOS ou Fedora

Avec yum, elle se fera en deux étapes :

- Activation des sources EPEL (Extra Package Enterprise Linux) :

```
$ sudo yum install epel-release
```

- Installation d'Ansible :

```
$ sudo yum install ansible
```

Au moment de l'écriture de ces lignes, Ansible n'est pas dans les sources officielles de Red Hat, d'où sa présence dans les sources EPEL. En revanche, vous disposerez de la dernière version stable.

4. Installation via pip

Dans le cas où le package que vous obtenez par votre système est trop ancien (ce qui peut arriver avec Debian par exemple) ou si vous voulez faire appel à un autre mécanisme de packaging (Gentoo, Slackware, etc.), il est également possible de passer par l'utilitaire pip (pip3 pour utiliser la version de Python 3).

Sous Debian, l'installation de pip3 se fait à l'aide de la commande apt suivie de l'option install et du nom du package python-pip3 :

```
$ sudo apt install python3-pip
```

Une fois pip installé, il est possible de lancer l'installation d'Ansible avec la commande suivante :

```
$ sudo pip3 install ansible
```

Dans le cas où vous auriez déjà une version installée que vous voudriez mettre à jour, il faudra ajouter l'option --upgrade :

```
$ sudo pip3 install --upgrade ansible
```

Sous Red Hat, il est également possible d'installer Ansible par pip. Pour cela, il faut :

Installer pip avec yum :

```
$ yum install python3-pip
```

Lancer l'installation d'Ansible avec pip :

```
$ pip3 install ansible
```

SOLYTI

5. Utilisation des packages releases

Dans certains cas, vous n'aurez pas la possibilité d'installer des packages sur la machine qui lance Ansible :

- Vous n'êtes pas administrateur de votre machine.
- Il est impossible d'installer des packages venant d'Internet.
- Ou tout simplement, vous souhaitez disposer de plusieurs versions d'Ansible sur une même machine.

Dans ce cas, vous serez heureux d'apprendre qu'il est tout à fait possible de disposer d'un interpréteur Ansible dans un répertoire quelconque.

Première chose, récupérer une copie du package du code source d'Ansible correspondant à la version qui vous intéresse. Ces fichiers sont disponibles à l'adresse suivante : http://releases.ansible.com/ansible/

L'exemple qui va suivre s'appuie sur la version 2.9.2 (ansible-2.9.2.tar.gz). L'installation aura lieu à la racine de l'utilisateur courant. L'archive est décompressée à l'aide de la commande tar suivie des options xfvz et du nom de l'archive :

```
$ tar xfv ansible-2.9.2.tar.gz
```

De là, vous disposez d'un répertoire ansible-2.9.2 avec tout le nécessaire.

Le lancement de l'interpréteur nécessite de réaliser quelques déclarations de variable dans l'environnement de l'utilisateur :

- ANSIBLE_VERSION : valeur de l'interpréteur (ici ansible-2.9.2) ;

- ANSIBLE_HOME : emplacement des fichiers de l'interpréteur Ansible ;
- MANPATH : emplacement des pages man Ansible (optionnel) ;
- PATH : emplacement des binaires Ansible (dans le sous-répertoire bin) concaténés avec l'ancienne valeur de la variable PATH ;
- PYTHONPATH : emplacement de la bibliothèque Ansible (dans le sous-répertoire lib).

Ci-dessous un exemple des déclarations à réaliser :

```
export ANSIBLE_VERSION="ansible-2.9.2"
export ANSIBLE_HOME="$HOME/$ANSIBLE_VERSION"
export MANPATH="$ANSIBLE_HOME/docs/man"
export PATH="$ANSIBLE_HOME/bin:$PATH"
export PYTHONPATH="$ANSIBLE_HOME/lib"
```

Ce fichier est présent dans le répertoire chapitre-01 avec les sources accompagnant ce livre sous le nom d'ansible-v2.9.2.

Une astuce que vous pouvez utiliser pour faire cohabiter plusieurs versions d'Ansible sur une même machine est de passer par des scripts contenant toutes ces instructions.

Quand vous souhaitez changer de version d'interpréteur, il suffit alors de « sourcer » le fichier associé à la version voulue. Ci-dessous un exemple d'utilisation avec le fichier ansible-v2.9.2 :

```
. ~/ansible-v2.9.2
```

6. Utilisation de virtualenv

Une dernière méthode d'installation d'Ansible (qui sera privilégiée par la suite) est de passer par l'outil virtualenv de Python. Ce mécanisme offre plusieurs avantages :

- Il permet de faire cohabiter très facilement plusieurs versions d'Ansible.
- Il utilise le mécanisme de packaging pip.
- Il permet de réaliser l'installation sans les droits administrateur.

En fonction du type de distribution, l'installation de virtualenv diffère légèrement. Ci-dessous la méthode pour RHEL/CentOS/Fedora :

```
$ yum install python3-virtualenv
```

Ci-dessous celle à utiliser pour l'installer sous Debian/Ubuntu :

```
$ apt install python3-virtualenv virtualenv
```

Ceci fait, il faut ensuite créer un environnement Python virtuel. Pour cela, lancez la commande virtualenv suivie des options suivantes :

- La version de l'interpréteur python 3 (--python=python3).
- Un emplacement de travail (/tmp/ansible par exemple).

Ci-dessous un exemple de lancement avec la version 3 de Python ainsi que le répertoire de travail /tmp/ansible :

```
$ virtualenv --python=python3 /tmp/ansible
```

Cet outil s'occupera alors de déposer une copie de travail indépendante dans le répertoire **/tmp/ansible**. Ci-dessous un exemple d'exécution de cette commande :

```
Running virtualenv with interpreter /usr/bin/python3
Using base prefix '/usr'
New python executable in /tmp/ansible/bin/python3
Also creating executable in /tmp/ansible/bin/python
Installing setuptools, pip, wheel...done.
.
```

SOLYTI

Ceci fait, reste à « activer » cet environnement à l'aide du script bin/activate présent dans le sous-répertoire de l'environnement virtuel. Pour cela, il est nécessaire de le sourcer dans l'environnement de l'utilisateur. Ci-dessous un exemple avec l'environnement se trouvant dans /tmp/ansible :

```
$ . /tmp/ansible/bin/activate
```

Un bon moyen de savoir si l'on utilise un environnement virtuel est de regarder où se trouve le binaire Python avec la commande type. Ci-dessous le résultat pour l'environnement /tmp/ansible :

```
$ type python
python is /tmp/ansible/bin/python
```

L'interpréteur Python est bien dans le répertoire /tmp/ansible.

L'installation d'Ansible quant à elle peut se faire avec pip comme vous l'avez vu dans la section Installation via pip.

7. Vérification de la version d'Ansible

Vous avez installé Ansible ? Vous pouvez maintenant vérifier la version à l'aide de la commande suivante :

```
$ ansible --version
```

Ci-dessous un exemple de sortie de cette commande (ici avec la version 2.9.2) :

```
ansible 2.9.2
  config file = None
  configured module search path = ['/root/...']
  ansible python module location = /tmp/.../site-packages/ansible
  executable location = /tmp/ansible/bin/ansible
  python version = 3.6.8 (default, May 21 2019, 23:51:36) [...]
```

Chapitre 3

Le protocole SSH

SOLYTI

1. À propos de SSH

Ansible est principalement conçu pour gérer des machines à l'aide du protocole SSH ou via des commandes lancées en local. Il est également possible de gérer d'autres types de machines, comme par exemple des systèmes Windows, des conteneurs Docker ou encore via des mécanismes d'isolation (chroot ou jail).

Même s'il est possible de passer par des mots de passe pour se connecter aux machines Linux, il est fortement recommandé de passer par des clés SSH. La suite sera consacrée à la génération des clés SSH et à leur propagation sur les machines à administrer.

2. Clé publique et clé privée

La première chose à faire si vous n'en avez pas sera de générer une clé. Cette dernière est constituée de deux parties : la clé privée et la clé publique. Une propriété importante vient du fait qu'il est très facile d'obtenir la clé publique depuis la clé privée, mais qu'il est très difficile d'obtenir la clé privée à partir de la clé publique.

La clé privée doit rester à tout prix secrète. En effet, elle permet à celui qui la possède de se connecter à vos machines.

La clé publique, quant à elle, peut être connue de tous étant donné qu'elle ne sert qu'à donner des droits de connexion à un serveur.

Ce couple de fichiers peut être comparé à un cadenas et sa clé : le cadenas ouvert représente la clé publique et la clé du cadenas représente la clé privée.

Un cadenas fermé protège quelque chose et ne peut être ouvert que si on dispose de la clé du cadenas. Enfin, il est plus simple d'ouvrir un cadenas avec sa clé plutôt qu'en le cassant.

Pour continuer avec cette analogie, tout le monde peut attacher un vélo en se servant d'un cadenas (même si on n'en possède pas la clé). En revanche, une fois le verrou posé, il sera beaucoup plus simple de l'ouvrir avec la clé plutôt que sans.

3. Génération de la clé

La génération de la clé est déclenchée par la commande ssh-keygen. On peut lui passer les options suivantes :

- Le type de clé à générer (rsa ou dsa) avec -t [rsa|dsa].
- L'emplacement où générer la clé avec -f <emplacement-clé>.

- Une passphrase (phrase secrète) pour protéger la clé avec l'option -N.
- Éventuellement la longueur de la clé (-b 2048 pour une clé de 2 048 bits).

Si vous ne passez pas de paramètres à la commande, ssh-keygen produira une clé RSA et la commande demandera les choses suivantes :

- l'emplacement de la clé (par défaut $HOME/.ssh/id_rsa) ;

- une passphrase et la confirmation de la passphrase.

4. Étapes de l'authentification

Il est important de comprendre comment fonctionne l'authentification par clé. Pour cela, la commande SSH procède à un certain nombre d'opérations :

- Vérification de la signature du serveur distant. Si ce dernier n'est pas connu, l'utilitaire ssh proposera de stocker la chaîne présentée par le serveur.

- Récupération des clés privées SSH présentes dans le répertoire .ssh (fichiers id_rsa ou id_dsa) de l'utilisateur et vérification des droits sur les fichiers.

- Présentation des clés aux serveurs distants. Si une clé correspond à une entrée dans le fichier ~/.ssh/authorized_keys distant, le serveur crée un challenge à résoudre par le client.

- Le client résout le challenge (c'est d'ailleurs à ce moment qu'il faut saisir la passphrase de la clé SSH) et le renvoie au serveur : l'utilisateur est authentifié.

5. Parc important de machines ou hébergement dans le cloud

Dans le cas de l'administration d'un parc important de machines (ou changeant souvent de clé SSH comme dans le cas de machines hébergées dans le cloud), il n'est pas faisable de maintenir la liste des signatures de machines distantes.

La désactivation de ce mécanisme se fait à l'aide des options SSH suivantes :

- Désactivation de la vérification stricte des clés SSH des machines distantes (StrictHostKeyChecking no).
- Stockage des signatures de machines dans le fichier /dev/null (UserKnownHostsFile /dev/null).

Cette configuration se fait en alimentant le contenu du fichier ~/.ssh/config. Ci-dessous le contenu de ce fichier avec ces deux options :

```
StrictHostKeyChecking no
UserKnownHostsFile /dev/null
```

6. Échange de clé par mot de passe

Dans la suite de l'exercice, les connexions SSH se feront avec l'utilisateur root. Dans le cas où il faudrait faire appel à un autre utilisateur (deploy, admin, etc.), faites l'échange de clé avec celui-ci. Le chapitre suivant sera consacré à gérer l'escalade de droit pour passer super utilisateur avec des mécanismes tel que sudo.

Par la suite, la machine rec-apache-1 sera administrée par clé SSH et vous disposerez du mot de passe de l'utilisateur root. Par conséquent, l'échange de clé se fera avec l'outil ssh-copy-id suivi du nom de la machine. Cet outil a pour fonction de prendre la clé publique SSH pour la déposer automatiquement sur la machine distante dans le fichier ~/.ssh/authorized_keys.

Il est également possible de faire précéder le nom de l'utilisateur avec lequel se connecter en utilisant une arobase ('@'). Ci-dessous un exemple d'échange de clé avec l'utilisateur root de la machine rec-apache-1 :

```
$ ssh-copy-id root@rec-apache-1
```

Dans le cas où la communication se ferait avec l'utilisateur deploy, la commande serait la suivante :

```
$ ssh-copy-id deploy@rec-apache-1
```

SOLYTI

Lors de la première connexion à une machine, ssh-copy-id demandera de faire confiance à la signature de cette dernière (sauf en cas de désactivation de cette vérification comme vu au cours du chapitre précédent) :

```
$ ssh-copy-id root@rec-apache-1
The authenticity of host 'rec-apache-1 (127.0.0.1)' can't be established.
ECDSA key fingerprint is SHA256:QPjmrxfQQERCRQ0SuYzBHvIdP+/1aOEQibnIkFoix2I.
Are you sure you want to continue connecting (yes/no)? yes
```

Le stockage de la signature se fait en entrant « yes ». Les lignes suivantes apparaissent pour indiquer la clé trouvée et la recopie qui va être réalisée :

```
Warning: Permanently added 'rec-apache-1' (ECDSA) to the list of
known hosts.
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed:
"/home/deploy/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new
key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if
you are prompted now it is to install the new keys
```

SOLYTI

Saisissez le mot de passe de l'utilisateur (ici root) :

```
root@rec-apache-1's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'root@rec-apache-1'"
and check to make sure that only the key(s) you wanted were added.
```

La clé est maintenant en place. Comme le message vous y invite, il est maintenant possible de se connecter avec la commande ssh <utilisateur>@rec-apache-1 pour s'assurer que l'échange de clé s'est bien passé :

```
$ ssh 'root@rec-apache-1'
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-79-generic x86_64)

* Documentation:  https://help.ubuntu.com
* Management:     https://landscape.canonical.com
* Support:        https://ubuntu.com/advantage

9 paquets peuvent être mis à jour.
0 mise à jour de sécurité.

Last login: Thu Jun  8 19:43:56 2017 from 127.0.0.1
```

7. Échange de clé sans mot de passe

L'échange de clé par mot de passe a été abordé. Le problème de cette méthode et qu'il n'est pas toujours possible d'avoir accès au sacro-saint mot de passe. Il est même probable que si vous l'obtenez, la configuration par défaut des machines empêchera la connexion directe par mot de passe (option PermitRootLogin prohibit-password dans le fichier /etc/ssh/sshd_config par exemple).

Dans ce cas, l'échange de clé devra être réalisé manuellement. Ci-dessous les opérations à dérouler dans pareil cas :

- Récupérez le contenu du fichier ~/.ssh/id_rsa.pub sur la machine Ansible.

- Connectez-vous sur la machine distante.

- Créez un répertoire .ssh avec des droits restreints à l'utilisateur :

```
$ mkdir -p ~/.ssh
$ chmod 700 ~/.ssh
```

- Créez un fichier authorized_keys avec également des droits restreints :

```
$ touch ~/.ssh/authorized_keys
chmod 600 ~/.ssh/authorized_keys
```

- Déposez la clé dans le fichier authorized_keys.

8. Gestion d'une passphrase avec Ansible

Comme évoqué plus tôt, il est possible de protéger sa clé avec une passphrase. En effet, si ce n'est pas le cas, une personne malveillante pourrait récupérer la clé et en faire ce qu'elle veut.

D'un autre côté, il faut communiquer la clé à Ansible ce qui implique de saisir la passphrase à chaque premier lancement. Attention de bien faire le ratio entre contrainte et sécurité.

Ci-dessous un exemple de lancement d'Ansible avec une clé SSH protégée par passphrase :

```
ansible -i rec-apache-1.inv -m ping rec-apache-1
Enter passphrase for key '/home/deploy/.ssh/id_rsa':
rec-apache-1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Pour éviter d'avoir à la saisir tout le temps, il est possible de passer par l'agent SSH de la session graphique (si vous utilisez KDE ou Gnome) ou en lançant un agent SSH (dans le cas où vous seriez connecté sur un serveur distant avec une session en mode texte).

Pour ajouter la clé à l'agent, il faut passer par la commande ssh-add suivie du chemin vers la clé SSH. La commande demandera alors de saisir la passphrase. Ci-dessous un exemple d'ajout de clé SSH :

```
$ ssh-add .ssh/id_rsa


Enter passphrase for .ssh/id_rsa:
Identity added: .ssh/id_rsa (.ssh/id_rsa)
```

Dans le cas où vous obtenez le message Error connecting to agent: No such file or directory, il vous faut lancer un agent SSH avec la commande eval $(ssh-agent). Ci-dessous un exemple de lancement de cet agent :

```
$ eval $(ssh-agent)
```

Cette commande doit retourner la sortie suivante :

```
Agent pid 20811
```

Dans le cas où Ansible est lancé depuis une application tierce, ces clés devront être gérées par un mécanisme externe. Dans le cas de Jenkins, le plugin SSH Agent Plugin peut être utilisé. Dans d'autres cas, il n'y aura pas forcément de solutions miracles. Il faudra alors se passer de ce mécanisme.

Chapitre 4

Utilisation de Ansible

SOLYTI

Ansible est maintenant installé et vos serveurs sont prêts à accepter des connexions SSH via un système de clés. Il est temps de commencer à administrer vos machines.

Par la suite, Ansible sera déjà installé et disponible dans le contexte de l'utilisateur. Les échanges de clés sont déjà réalisés avec les serveurs distants.

Ansible en mode ad hoc

Ansible est maintenant disponible et prêt pour les premiers tests. Le mode que vous verrez en premier est celui dans lequel chaque opération doit être spécifiée unitairement.

Ce mode convient parfaitement pour s'assurer que la communication se passe bien ou pour réaliser des opérations simples, comme par exemple la création d'un répertoire ou d'un utilisateur.

1. Création d'un fichier d'inventaire

Mais avant de partir sur le lancement d'Ansible sur des machines, il faut créer un fichier d'inventaire avec les machines à administrer. Ce fichier peut être écrit dans plusieurs formats comme par exemple :

- format INI
- format YAML

Le format INI est le format historique des inventaires Ansible. Dans les inventaires écrits avec ce format, chaque machine occupe une ligne. Ces machines peuvent également être regroupées à l'aide de sections. Ces aspects seront vus un peu plus loin.

Dans le cas des inventaires au format YAML, les machines sont regroupées dans des structures spécifiques qui seront vues plus loin dans le chapitre de présentation du format YAML.

Pour démarrer, créez un fichier rec-apache.inv (disponible dans les fichiers d'exercice). Ce dernier va contenir une seule ligne avec le nom de la machine rec-apache-1. En plus du nom de la machine, ajoutez la déclaration ansible_user=root (sur la même ligne). Vous indiquez ainsi que la connexion sur la machine se fera avec l'utilisateur root. Ci-dessous le contenu de ce fichier :

```
rec-apache-1 ansible_user=root
```

Pour tester la communication, vous pouvez appeler la commande ansible avec les options suivantes :

- le module ping (-m ping) ;
- le fichier inventaire (-i rec-apache-1.inv) ;
- le groupe sur lequel vous souhaitez travailler (ici all pour désigner toutes les machines).

Ci-dessous la commande correspondante :

```
$ ansible -i rec-apache-1.inv -m ping all
```

Si la communication se passe bien, vous devriez obtenir le message suivant :

```
rec-apache-1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

La communication se passe bien, passez à la suite.

2. Utilisateur non root

Vous avez travaillé avec l'utilisateur root. Si toutefois vous n'avez pas le droit de vous connecter directement avec, il faudra le préciser à Ansible. Pour cela, changez la valeur de la variable ansible_user associée à la machine.

Reprenez l'inventaire rec-apache-1.inv. Afin de permettre la connexion avec l'utilisateur deploy, le fichier devra contenir la déclaration suivante :

```
rec-apache-1 ansible_user=deploy
```

Pour vérifier qu'Ansible emploie bien cet utilisateur, vous pouvez faire appel au module setup. Pour mémoire, ce module s'occupe de récupérer un ensemble de variables qui va caractériser la machine sur laquelle vous travaillez (adresses DNS, IP, points de montage, etc.). Dans le mode de fonctionnement à base de playbook que vous verrez plus loin, ce module est associé à l'option gather_facts.

Parmi ces variables, ansible_user_id permet de récupérer le nom de l'utilisateur avec lequel vous vous connectez (root, deploy, etc.). Afin de ne pas obtenir une sortie trop longue, vous allez filtrer le résultat sur cette seule variable.

Pour lancer cette opération, passez les options suivantes à la commande ansible :

-m setup : lancement du module setup ;

-a filter=ansible_user_id : filtre sur la variable ansible_user_id ;

-i inventaire : inventaire à utiliser ;

all : nom des machines à sélectionner.

Ci-dessous la commande correspondante :

```
ansible -i rec-apache-1.inv -m setup \
        -a 'filter=ansible_user_id'  all
```

Ci-dessous le résultat de cette commande :

- Cas où la variable ansible_user est positionnée sur l'utilisateur root (ansible_user=root) :

```
rec-apache-1 | SUCCESS => {
    "ansible_facts": {
        "ansible_user_id": "root"
    },
    "changed": false
}
```

SOLYTI

Cas où ansible_user est positionnée sur l'utilisateur deploy (ansible_user=deploy) :

```
rec-apache-1 | SUCCESS => {
    "ansible_facts": {
        "ansible_user_id": "deploy"
    },
    "changed": false
}
```

3. Utilisateur SSH non root et mécanisme sudo

Par la suite, l'échange de clés a été fait avec le compte deploy sur la machine rec-apache-1.

Vous arrivez à communiquer avec vos machines distantes, mais vous n'avez pas forcément les droits suffisants pour faire ce que vous voulez (ajout de packages, suppression de fichiers système).

Un mécanisme courant de sécurisation des serveurs est la connexion avec un utilisateur sans privilège puis l'escalade à l'aide de l'outil sudo. Si vous êtes utilisateur de la distribution Ubuntu, vous avez l'habitude de travailler de la manière suivante :

- Vous travaillez avec un utilisateur à votre nom (yannig, sarah ou olivier) qui n'a pas le droit de tout faire.

- Lorsque vous voulez administrer votre poste (installation de packages par exemple), vous passez root à l'aide de sudo.

Ansible est compatible avec ce mode de fonctionnement.

Première chose à vérifier, l'échange de clés SSH avec la commande ssh suivante :

```
$ ssh deploy@rec-apache-1
```

Si tout se passe bien, la commande devrait vous renvoyer un prompt sur la machine distante. De là, vous allez pouvoir observer la présence de droits sudo attachés à l'utilisateur avec la commande sudo -l :

```
$ sudo -l
```

Si le système vous demande le mot de passe de l'utilisateur, c'est que ce dernier n'a aucun droit…

Afin de corriger ce point, vous allez devoir ajouter la ligne suivante dans le fichier /etc/sudoers :

```
deploy ALL=(ALL:ALL) NOPASSWD: ALL
```

SOLYTI

Une manière traditionnelle de faire les choses serait de s'authentifier en tant que root avec la commande su et de réaliser les manipulations avec vi. Mais comme vous avez dû vous en rendre compte, vous avez entre les mains un manuel sur le fonctionnement d'Ansible et non sur de l'administration système.

Dans pareil cas, il est possible d'utiliser Ansible pour effectuer cette configuration.

- Pour cela, vous allez changer le mode d'escalade par défaut (à base de sudo) pour passer par le mécanisme « su ». Vous verrez qu'il est même préférable de procéder de cette manière pour éviter certaines erreurs.

Pour cela, vous passerez par le module lineinfile (option -m lineinfile). Outre le nom du module, vous devrez passer d'autres arguments et notamment :

- Les arguments (entre double quote) du module avec l'option -a. Dans ces arguments, vous retrouverez :

  - le fichier à modifier avec l'option path=/etc/sudoers ;

- la ligne à ajouter, ici line=’deploy ALL=(ALL:ALL) NOPASSWD: ALL’ ;

- comme vous éditez un fichier sudo, il est possible de le valider avec l’option validate=’visudo -cf %s’ ;

- Comme vu un peu plus tôt, vous allez indiquer que vous souhaitez passer en tant que root avec la méthode su. Il s'agit là de l'option --become-method=su.

- L'option --become (-b en version courte) indiquera à Ansible de faire une escalade (par défaut en tant que root).

- Enfin, vous allez demander à l'utilisateur de saisir le mot de passe au moment du lancement d'Ansible avec l'option --ask-become-pass (ou -K pour la version courte).

Si vous reprenez toutes ces options, vous allez donc lancer la commande suivante :

```
ansible -i rec-apache-1.inv -m lineinfile \
 -a "path=/etc/sudoers \
 line='deploy ALL=(ALL:ALL) NOPASSWD: ALL'" \
 --become-method=su --become --ask-become-pass all
```

La première chose que va réaliser Ansible sera de vous demander le mot de passe de l'utilisateur root avec le message suivant :

```
SU password:
```

Une fois le mot de passe saisi, Ansible va faire le nécessaire et devrait vous renvoyer le résultat suivant :

```
rec-apache-1 | SUCCESS => {
    "backup": "",
    "changed": true,
    "msg": "line added"
}
```

SOLYTI

Une question légitime à poser suite à cette opération serait la relative complexité de la commande. En effet, la même opération réalisée directement à la main aurait été aussi rapide.

Imaginez une infrastructure serveur avec 10, 100, voire 1 000 machines. Si ces 1 000 machines ont toutes le même mot de passe root, vous allez pouvoir lancer cette modification sur les 1 000 machines en même temps.

La seule différence : votre fichier d'inventaire. Ceci est un exemple assez simple, mais qui rend bien compte de l'intérêt d'un outil comme Ansible.

Assurez-vous que la modification a bien été apportée en vous reconnectant sur la machine distante avec ssh et en lançant la commande sudo -l :

```
$ sudo -l
Entrées par défaut pour deploy sur travail :
   env_reset, mail_badpass, secure_path=/usr/local/[...]:/bin

L'utilisateur deploy peut utiliser les commandes suivantes sur travail :
   (ALL : ALL) NOPASSWD: ALL
```

SOLYTI

La dernière ligne indique bien que vous avez le droit de lancer toutes les commandes sans saisir de mot de passe.

Pour vous assurer que tout fonctionne, vous pouvez relancer la commande ansible avec les options suivantes :

- le module setup (-m setup) ;

- le filtrage sur la variable ansible_user_id (filter=ansible_user_id) ;

- l'option permettant de passer en tant que root (--become).

Cette commande vous permettra de savoir quel est l'utilisateur avec lequel vous travaillez. Ci-dessous la commande correspondante :

```
$ ansible -i rec-apache-1.inv -m setup \
        -a 'filter=ansible_user_id' --become all
```

Cette commande devrait vous renvoyer les lignes suivantes :

```
rec-apache-1 | SUCCESS => {
    "ansible_facts": {
        "ansible_user_id": "root"
    },
    "changed": false
}
```

SOLYTI

Pour lancer vos opérations avec l'utilisateur apache à la place de root, ajoutez l'option --become-user apache à la commande précédente :

```
ansible -i rec-apache-1.inv -m setup \
        -a 'filter=ansible_user_id' \
        --become-user apache --become all
```

Ci-dessous le résultat de cette commande :

```
rec-apache-1 | SUCCESS => {
    "ansible_facts": {
        "ansible_user_id": "apache"
    },
    "changed": false
}
```

Cette fois-ci, vous lancez bien vos commandes en tant qu'utilisateur apache.

SOLYTI

Les autres outils d'Ansible : playbook et doc

Jusqu'à maintenant, vous vous êtes surtout servi d'Ansible en mode ad hoc pour vérifier la communication avec vos serveurs distants (module setup et ping) et éventuellement modifier le contenu d'un fichier (lineinfile).

Vous allez maintenant voir qu'il existe plusieurs utilitaires avec Ansible. Par la suite, vous verrez le mode playbook qui sera à privilégier la plupart du temps. Il existe également l'utilitaire ansible-doc qui sera également abordé.

Ce dernier vous offre la possibilité de consulter la documentation des différents modules sans avoir à vous connecter sur Internet sur le site officiel d'Ansible.

Reprenez la machine de recette apache rec-apache-1. Comme vous arrivez à communiquer avec cette machine et à réaliser une escalade en tant que root, vous allez y faire les opérations suivantes :

- installation du package apache ;
- activation et démarrage du service apache ;
- recopie d'un fichier à la racine du serveur.

Cette machine se base sur une distribution à base de RPM (Red Hat, Fedora ou CentOS). Dans ces cas-là, il faut appeler le module yum pour procéder à l'installation (dans le cas d'une distribution dérivée de Debian, il aurait fallu utiliser le module apt). Vous passerez en paramètre le nom du package à installer (-a name=httpd). Par défaut, le module installera le package, mais il est possible d'expliciter l'opération en ajoutant l'option state=present ou state=installed.

Pour connaître toutes les options d'un module, vous pouvez faire appel à la commande ansible-doc suivie du nom du module. Pour lister les modules présents dans Ansible, vous pouvez utiliser la commande ansible-doc -l.

En ajoutant l'inventaire (-i inventaire), la liste des machines à installer (all) ainsi que l'option permettant de passer root (-b), la commande à lancer sera la suivante :

```
$ ansible -i rec-apache-1.inv -b -m yum -a name=httpd all
```

Ci-dessous le résultat de votre commande :

```
rec-apache-1 | SUCCESS => {
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
        "Loaded plugins: [...]]
}
```

Passez maintenant au démarrage du service avec le module du même nom. Un rapide coup d'œil avec ansible-doc vous indique que vous devrez utiliser les options suivantes :

- le nom du service (name=httpd) ;
- l'état attendu du service (state=started) ;
- l'activation du service au démarrage (enabled=yes).

La commande suivante reprend les options précédentes :

```
$ ansible -i rec-apache-1.inv -b -m service \
   -a "name=httpd state=started enabled=yes" all
```

Ci-dessous le résultat de cette exécution :

```
rec-apache-1 | SUCCESS => {
   "changed": true,
   "enabled": true,
   "name": "httpd",
   "state": "started",
   "status": {...}
}
```

SOLYTI

Autre opération à réaliser dans le cas de la présence d'un pare-feu : permettre la communication de l'extérieur vers le port HTTP. Cette opération se fait à l'aide du module firewalld suivi des options suivantes :

- Le nom du service à autoriser (service=http).
- Le champ permanent à la valeur yes pour conserver le réglage au démarrage.
- Le champ state à la valeur enabled.

SOLYTI

Ci-dessous la commande complète correspondante :

```
$ ansible -i rec-apache-1.inv -b -m firewalld \
    -a "service=http permanent=yes state=enabled" all
```

Et le résultat de cette dernière :

```
apache1 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": true,
    "msg": "Permanent operation, Changed service http to enabled"
}
```

Enfin, vous allez copier un fichier test.html à la racine du serveur nouvellement démarré. Ci-dessous le contenu du fichier test.html :

```
$ cat test.html
Copie de fichier par Ansible
```

Pour votre copie, vous ferez appel au module copy. Ce dernier va avoir besoin d'un certain nombre d'options pour réaliser les opérations souhaitées :

- L'emplacement du fichier source (src=test.html)
- Le répertoire destination (dest=/var/www/html)
- Le propriétaire du fichier (owner=apache)
- Le groupe du fichier (group=apache)
- Le mode de protection du fichier (mode=644)

Encore une fois, en reprenant les options de sécurité, vous aurez à lancer la commande suivante :

```
$ ansible -i rec-apache-1.inv -b -m copy \
   -a "src=test.html owner=apache group=apache \
    dest=/var/www/html" all
```

Ci-dessous le résultat correspondant à cette commande :

```
rec-apache-1 | SUCCESS => {
   "changed": true,
   "checksum": "d1d1d52701e88f413b651e5c4b1ff6fc9e226c1f",
   [...]
}
```

Lancez maintenant un navigateur et accédez au fichier test.html à la racine du serveur Apache nouvellement installé (http://rec-apache-1/test.html) :

Mission accomplie : le navigateur renvoie bien le contenu du fichier copié par Ansible.

Quelques notions sur le format YAML

Avant d'aller plus loin dans la découverte d'Ansible, il faut connaître quelques notions sur le format YAML. En effet, ce format est utilisé par Ansible pour énormément de choses :

- les fichiers de variables (nom d'utilisateur, mot de passe, etc.) ;

- la description des opérations à réaliser (notamment avec les playbooks) ;
- les inventaires et la configuration d'Ansible (depuis la version 2.4).

Pour l'essentiel, YAML permet d'écrire des structures de données. Ces données peuvent être spécifiées sous la forme de listes ou sous la forme de dictionnaires.

Ce langage propose également plusieurs façons d'écrire les mêmes structures en fonction de ce que vous souhaitez mettre en avant : lisibilité ou compacité de votre description (même si généralement, il faut privilégier la lisibilité).

À noter que la notation au format JSON permet le même type de stockage (Ansible l'accepte également).

Ces deux langages partagent énormément de caractéristiques communes étant donné que YAML est une extension du langage JSON. La principale différence entre JSON et la notation YAML réside surtout au niveau de la lisibilité.

1. Déclaration de variables simples

Le langage YAML sert à déclarer des variables. Par défaut, il n'y a aucune contrainte au niveau du nom des variables si ce n'est qu'il doit y avoir un nom suivi d'un deux-points (:) suivi d'un espace (' '). En revanche, Ansible n'accepte pas n'importe quelle déclaration.

Un nom de variable valide devra impérativement être composé d'une suite de caractères alphanumériques et/ou de tirets bas (_). Le premier caractère devra toujours être un caractère alpha ou un tiret bas :

- exemple de variable valide pour Ansible : _42, var1, a_b ;
- exemple de variable valide pour YAML, mais inutilisable pour Ansible : 42a, a-b, a%b.

Après la déclaration de la variable, vous trouverez son contenu. Par défaut, si vous ne mettez pas de guillemets, YAML cherchera à typer la variable (entier ou à virgule flottante). Ci-dessous quelques exemples de déclarations de variables :

```
---
# Un fichier yaml démarre par les 3 tirets ci-dessus
# Déclaration simple
chaine_simple: "Une chaîne simple"
# Ici _42 va contenir un entier :
_42: 42
# _33 va contenir un chiffre à virgule :
_33: 33.333
```

2. Les tableaux en YAML

Premier type de structure : le tableau. Le nom du tableau doit respecter les mêmes contraintes que pour une variable simple. Il est ensuite possible de lister les éléments avec des tirets (-) précédés d'un espace (ou plus). Chaque tiret sera une valeur que vous voulez voir dans votre tableau.

Une autre notation existe : cette dernière se rapproche de JSON dans lequel vous encadrez ces valeurs avec des crochets ([et]).

Prenez le cas du tableau a qui contient les éléments 1, 2 et « trois ». Ci-dessous la notation longue :

```
a:
 - 1
 - 2
 - "trois"
```

Le nombre d'espaces devant le tiret (-) est laissé au soin de la personne en charge de déclarer le fichier. En revanche, une fois que vous aurez choisi ce nombre d'espaces, il vous faudra le respecter pour tout le tableau. Il y a également interdiction formelle de mélanger les espaces et les tabulations.

En version compacte, cette déclaration pourra se faire de la manière suivante :

```
# Version compacte
a: [ 1, 2, "trois" ]

# Version très compacte
a: [1,2,"trois"]

# Version invalide (manque l'espace après ':')
a:[1,2,"trois"]
```

À noter que dans Ansible, l'accès au contenu de la variable a pourra se faire avec son nom suivi de l'indice de l'élément à obtenir entre crochets ([et]), le premier élément étant stocké à l'indice 0.

Dans votre cas, l'accès au chiffre 1 se fera avec l'expression a[0] tandis que l'accès à la chaîne « trois » se fera avec a[2].

3. Les structures clé/valeur

Un autre type de structure couramment utilisé en YAML est le tableau de hachage. Il s'agit d'associer un champ (un peu à l'image d'un nom de variable) avec une valeur (qui peut être également une valeur simple, un tableau ou encore une structure clé/valeur). Avec ce mécanisme, il est possible de représenter n'importe quelle structure arborescente.

Comme pour le tableau simple, la déclaration commence par un nom de variable suivi de deux-points (:) suivi d'un espace. Les différents champs sont ensuite ajoutés à la ligne en respectant le même nombre d'espaces. Prenez le cas d'un utilisateur avec les champs « nom » et « prenom ». Cette déclaration se fait de la manière suivante :

```
utilisateur1:
 nom: perre
 prenom: yannig

utilisateur2:
 nom: perre
 prenom: sarah
```

Sur l'utilisateur utilisateur1, vous souhaitez ajouter le champ date_de_naissance. Ce champ contiendra les sous-champs jour, mois et annee. Les sous-champs seront rattachés à date_de_naissance en les déclarant en dessous du champ date_de_naissance et en augmentant l'indentation.

Ci-dessous la déclaration de l'utilisateur utilisateur1 avec la date de naissance rattachée :

```
utilisateur1:
 nom: perre
 prenom: yannig
 date_de_naissance:
   jour: 7
   mois: 11
   annee: 1977
```

Comme pour les tableaux, il est possible d'utiliser une notation compacte. Ci-dessous la variable utilisateur2 dans une version plus courte :

```
utilisateur2: { nom: perre, prenom: sarah }
```

4. Tableau de tables de hachage

Le tableau de tables de hachage est la dernière structure dont vous aurez besoin. La déclaration se fait de la même manière que pour le tableau, mais en remplaçant les éléments par des tables de hachage.

Dans l'exemple qui suit, vous allez déclarer une liste d'utilisateurs constituée des deux utilisateurs de tout à l'heure :

```
liste_utilisateurs:
 - nom: perre
   prenom: yannig
 - nom: perre
   prenom: sarah
```

Comme avec les tables de hachage, il est possible d'utiliser une version compacte :

```
# Version un peu plus compacte
users:
 - { nom: perre, prenom: yannig }
 - { nom: perre, prenom: sarah }
# Version plus compacte, mais moins lisible
users: [{nom: perre, prenom: yannig},{nom: perre, prenom: sarah}]
```

5. Inventaire au format YAML

Les inventaires au format YAML sont supportés depuis la version 2.4 d'Ansible. Auparavant, ces fichiers devaient forcément être écrits au format INI.

Le choix du format YAML a été dicté par plusieurs raisons :

- Uniformisation des types de fichiers dans Ansible.
- Généralisation du mécanisme de gestion des inventaires.

Ces deux aspects seront approfondis plus loin avec l'introduction des notions de fichiers de variables ou lors de l'écriture de fichiers d'inventaires dynamiques.

Pour en revenir au format de fichier YAML, les machines doivent être stockées dans une structure répondant aux critères suivants :

- Un nom de groupe (par défaut all).

- Une table de hachage hosts contenant un enregistrement par machine.

- Chaque machine étant elle-même une table de hachage contenant l'affectation des variables à cette machine.

Pour reprendre l'exemple d'inventaire du chapitre précédent, ce dernier pourra s'écrire au format YAML sous la forme suivante :

```
all:
  hosts:
    rec-apache-1:
      ansible_user: root
```

Introduction de la notion de playbook

La suite du chapitre sera consacrée à la ré-écriture de l'installation d'Apache de tout à l'heure. Les quatre commandes à passer étaient relativement complexes :

- Passage des options de sécurité.

- Options longues et nombreuses pour faire fonctionner les modules (utilisation de double quote pour les options et séparation à l'aide d'espaces).

- Enchaînement manuel des commandes

Toutes ces pratiques vont à l'encontre de la création de procédures simples et rapides facilement réutilisables. Autre problème, ces aspects ne sont pas implicites alors qu'il serait intéressant de les stocker quelque part (Git par exemple).

C'est là qu'intervient la notion de playbook pour répondre à ce type de situation.

Un playbook est un fichier au format YAML. Ce dernier va donner une liste d'instructions. Ces instructions sont passées à Ansible dans l'ordre de leur déclaration. L'avantage par rapport au mode ad hoc est que vous aurez ainsi tout décrit dans un fichier, y compris l'enchaînement des opérations.

1. Structure d'un playbook

Un playbook peut contenir énormément d'informations, mais la plupart du temps vous pouvez vous contenter d'un sous-ensemble relativement restreint. Ci-dessous, une liste des champs auxquels vous ferez forcément appel :

- le nom de playbook (champ name) ;

- la liste de machines sur laquelle vous allez travailler (champ hosts) ;

- des options de gestion de la sécurité (optionnelles) ;

- la gestion du lancement de la collecte des informations (setup) constituant la machine (champ gather_facts) ;

- enfin, une liste d'instructions à dérouler (tasks).

Dans le cas du playbook d'installation d'Apache, la liste d'instructions comportera :

- un appel au module yum pour l'installation du package ;
- un appel au module service ;
- un appel au module firewalld ;
- enfin, un appel à copy.

Pour chaque module, il faudra :

- le nom du module à exécuter ;

- les options nécessaires au fonctionnement du module dans des sous-champs ;

- enfin, pour des raisons de clarté, il est possible d'ajouter le champ name avec une description de l'opération réalisée.

Ci-dessous un exemple de playbook qui permettra de réaliser l'installation d'Apache :

```
- name: "Apache Installation"
  hosts: all
  tasks:
    - name: "Install apache package"
      yum:
        name: "httpd"
        state: "present"
    - name: "Start apache service"
      service:
        name: "httpd"
        state: "started"
        enabled: yes
    - name: "Allow http connections"
      firewalld:
        service: "http"
        permanent: yes
        state: "enabled"
    - name: "Copy test.html"
      copy:
        src: "test.html"
        dest: "/var/www/html"
       owner: "apache"
       group: "apache"
```

2. Lancement d'un playbook

Le playbook est maintenant prêt, il est temps de le lancer à l'aide de la commande ansible-playbook. En plus de l'appel à cette commande, les options suivantes devront être passées en paramètre :

- l'inventaire rec-apache-1.inv (-i rec-apache-1.inv) ;
- le playbook à lancer (install-apache.yml) ;
- l'option become (--become ou -b) ;

Il n'est plus nécessaire de passer le groupe de machines (all). En effet, le playbook contient un champ hosts qui va préciser la liste des machines sur lesquelles se dérouleront les opérations.

Ci-dessous, la commande à lancer :

```
$ ansible-playbook -b -i rec-apache-1.inv install-apache.yml
```

Et ci-dessous le résultat du lancement :

```
PLAY [Apache installation] ***********************************

TASK [Gathering Facts] ***************************************
ok: [rec-apache-1]

TASK [Installation package apache] ***************************
ok: [rec-apache-1]

TASK [Start apache service] **********************************
ok: [rec-apache-1]

TASK [Allow http connections] ********************************
ok: [rec-apache-1]

TASK [Copy test.html] ****************************************
ok: [rec-apache-1]

PLAY RECAP ***************************************************
rec-apache-1    : ok=4    changed=0    unreachable=0    failed=0
```

Comme les différentes opérations présentes dans le playbook avaient déjà été lancées indépendamment sur la machine rec-apache-1, aucune opération n'a entraîné de changement sur le système. À noter qu'il s'agit d'une des caractéristiques d'Ansible : la réentrance[1]

[1]En informatique, la réentrance est la propriété pour une fonction d'être utilisable simultanément par plusieurs tâches utilisatrices. La réentrance permet d'éviter la duplication en mémoire vive d'un programme utilisé simultanément par plusieurs utilisateurs.

Structure d'un inventaire

L'inventaire sous Ansible est un fichier au format INI. La déclaration d'un groupe se fait en utilisant le nom du groupe entre crochets. Les machines rattachées à ce groupe sont simplement ajoutées à la suite de la déclaration du groupe (une machine par ligne).

Prenez un exemple avec les éléments suivants :

- un groupe apache constitué de la machine rec-apache-1 ;
- un groupe mysql avec la machine rec-mysql-1.

Ci-dessous un fichier d'inventaire correspondant à cette déclaration :

```
[apache]
rec-apache-1

[mysql]
rec-mysql-1
```

À noter qu'il est également possible de stocker cette déclaration sous la forme d'un fichier YAML. Dans ce cas, la déclaration prendra la forme suivante :

```
apache:
  hosts:
    rec-apache-1: {}

mysql:
  hosts:
    rec-mysql-1: {}
```

Par la suite, ce fichier sera référencé sous le nom de fichier host ou fichier d'inventaire. Il sera accessible sous le nom de recette.inv (au format INI) ou recette.yml (format YAML).

Pour y faire appel, il faudra passer l'option -i à ansible avec le nom du fichier.

Afin de vérifier que la communication avec les machines distantes se passe bien, vous allez appeler ansible avec le module ping (en passant le nom du module avec l'option -m).

Ci-dessous un exemple d'appel avec l'inventaire YAML :

```
$ ansible -i recette.yml -m ping all
```

Si tout se passe bien, Ansible devrait renvoyer le résultat suivant :

```
rec-mysql-1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
rec-apache-1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

SOLYTI

1. Groupes par défaut

En plus des groupes que vous déclarez dans votre inventaire, il existe deux autres groupes : all et ungrouped. Le premier contient toutes les machines de l'inventaire sans distinction. Les machines qui ne sont rattachées à aucun groupe (c'est le cas pour localhost si vous ne le déclarez pas dans votre inventaire) se retrouveront dans ungrouped.

Un moyen simple de consulter cette liste est de faire appel à la commande ansible avec le module debug (-m debug) en lui passant la variable groups (-a var=groups).

Ci-dessous, la commande à lancer :

```
$ ansible -m debug -a var=groups localhost
```

Cette commande vous renvoie alors le résultat suivant :

```
[WARNING]: provided hosts list is empty, only localhost is available

localhost | SUCCESS => {
    "changed": false,
    "groups": {
        "all": [],
        "ungrouped": [
            "localhost"
        ]
    }
}
```

2. Mode de connexion aux machines

Par défaut, vous accédez aux machines avec SSH. Néanmoins, Ansible gère d'autres types de connexion. Vous pouvez retenir les modes suivants :

- ssh : connexion vers tout type d'Unix (Linux, *BSD, AIX, Solaris, etc.) ;

- local : cas de la machine localhost ;

- docker : connexion à des conteneurs ;

- chroot/jail : travail dans un environnement isolé ;

- winrm : connexion aux machines Windows.

Il existe de très nombreux plugins. Le mieux est de se référer à la documentation d'Ansible.

Dans le cas où vous voulez changer le mode de connexion par défaut à votre machine, vous devez renseigner la variable ansible_connection. Pour Windows, le mode devra être positionné sur la valeur ansible_connection=winrm et pour un conteneur docker, vous utiliserez ansible_connection=docker. Ce changement de mode peut se faire en ajoutant la variable directement après la déclaration de la machine.

Ci-dessous un exemple de déclaration d'une machine Windows et d'un conteneur Docker :

```
[active-directory]
active-directory-1 ansible_connection=winrm

[container]
container-1 ansible_connection=docker
```

Enfin, il est également possible de changer temporairement le mode de connexion à une machine au niveau d'un playbook en utilisant l'instruction connection.

Ci-dessous un exemple de playbook utilisant le mode local :

```
- name: "Retrieve facts and store them locally"
 hosts: all
 gather_facts: no
 connection: local
 tasks:
   - name: "Create directory for each machine"
     file:
       path: "/tmp/{{inventory_hostname}}"
       state: "directory"
```

SOLYTI

3. Regroupement de machines

Il est possible de regrouper des machines à l'aide de sections se terminant par ":children" (exemple : linux:children). Ce mécanisme permet de regrouper des machines dans des ensembles plus grands et d'éviter de les redéclarer.

Reprenez le cas de l'inventaire recette.inv/recette.yml et ajoutez-y de nouvelles machines :

- La machine active-directory-1 dans le groupe active-directory.

- Le conteneur container-1 dans le groupe microservices.

Vous allez également regrouper les machines de la manière suivante :

- Un groupe linux contenant les machines du groupe apache et mysql.
- Dans le groupe windows les machines du groupe active-directory.
- Enfin un groupe container avec les machines du groupe microservices.

Vous profiterez de ces groupes pour affecter le mode de connexion à l'aide de sections groupe:vars (variable ansible_connection positionnée à winrm ou docker en fonction du type d'OS).

Ci-dessous le fichier d'inventaire correspondant au format INI :

```
[all:vars]
ansible_connection=local

[apache]
rec-apache-1 apache_url=rec.wiki.localdomain

[mysql]
rec-mysql-1 mysql_user_password=MyPassWord!

[active-directory]
active-directory-1

[microservices]
container-1 ansible_connection=docker

[linux:children]
apache
mysql

[windows:children]
active-directory

[container:children]
microservices

[windows:vars]
ansible_connection=winrm

[container:vars]
ansible_connection=localhost
```

Cette déclaration peut également prendre la forme d'une structure YAML en respectant les adaptations suivantes :

- un champ par groupe à déclarer
- rattachement des machines dans le champ hosts
- rattachement des déclarations de variables dans le champ vars

- possibilité de composer des groupes à l'aide du champ children

- affectation de variables aux machines en ajoutant des sous-champs au niveau de leur déclaration

Ainsi, la même déclaration au format YAML prendra la forme suivante :

```
all :
  vars:
    ansible_connection: local

linux:
  children:
    apache:
      hosts:
        rec-apache-1:
          apache_url: “rec.wiki.localdomain”

    mysql:
      hosts:
        rec-mysql-1:
          mysql_user_password: “MyPassWord!”

windows:
  children:
    active-directory:
      hosts:
        active-directory-1: {}
  vars:
    ansible_connection: “winrm”

container:
  children :
    microservices :
      hosts :
        container-1 :
          ansible_connection :  "docker"
      vars :
        ansible_connection :  "localhost"
```

Dans le cas où la machine n'aurait pas de variable rattachée, prenez l'habitude de la déclarer en tant que tableau de hachage vide (machine : {}). Cette habitude évitera certaines erreurs de déclaration.

4. Variables d'inventaire

Vous avez vu comment déclarer vos machines et comment changer le mode de connexion à celles-ci avec ansible_connection. Vous allez maintenant voir comment pousser le mécanisme un peu plus loin. En effet, vous aurez parfois envie de personnaliser vos machines en fonction de variables.

Le nom de ces variables doit être forcément en alphanumérique et commencer par une lettre ou un tiret-bas (cf. Quelques notions sur le format YAML du chapitre Utilisation d'Ansible).

Ces variables peuvent être positionnées au niveau de la déclaration des machines ou au niveau des groupes. Ci-dessous un exemple de déclaration de la variable apache_url pour la machine rec-apache-1 :

```
[apache]
rec-apache-1 apache_url=rec.wiki.localdomain
```

Cette déclaration prendra la forme suivante au format YAML :

```
apache:
  hosts:
    rec-apache-1:
      apache_url: "rec.wiki.localdomain"
```

La déclaration au niveau du groupe se fera à l'aide d'une section groupe:vars.

Ci-dessous un exemple de déclaration :

```
[mysql:vars]
mysql_user_password=MyPassWord!
```

SOLYTI

Et la même déclaration au format YAML :

```
mysql:
  vars:
    mysql_user_password: "MyPassWord!"
```

Il existe également d'autres moyens de rattacher des déclarations de variables à une machine ou à un groupe de machines : l'utilisation des répertoires group_vars et host_vars.

Le répertoire group_vars contient des fichiers portant le même nom que les groupes de votre inventaire. Même chose pour ceux de host_vars, qui porteront le même nom que les machines de votre inventaire. Ces fichiers peuvent éventuellement se terminer avec l'extension .yml. Ils doivent suivre impérativement les formats YAML ou JSON.

Il est possible de créer des fichiers de variables (au format YAML) ou encore de passer à Ansible des variables à l'aide de l'option -e :

```
$ ansible -e variable=valeur -e @fichier-variables.yml \
          -m debug -a var=variable localhost
```

5. Hiérarchie des variables

Vous avez vu qu'il était possible d'attacher des variables par plusieurs moyens. Il faut maintenant savoir dans quel ordre ces variables sont appliquées (par ordre inverse de priorité) :

- variables du groupe dans le fichier host ;
- variables du groupe dans les fichiers group_vars ;

- variables de la machine au niveau du fichier host ;
- variables de la machine au niveau du fichier host_vars ;
- variables se trouvant dans un fichier YAML (-e @fichier.yml) ;
- variables directement passées à Ansible (-e variable=valeur).

À noter qu'il existe également d'autres variables :

- Les variables « magiques » d'Ansible (emplacement du fichier inventaire, nom de la machine dans l'inventaire, liste des groupes, etc.).

- Les variables de découverte de la machine (ansible_user_id, ansible_user_shell, ansible_system, etc.). Ces variables sont récupérées au moment de la tâche setup.

- Les valeurs par défaut des rôles.

- Les variables des rôles.

Dans le répertoire variables des fichiers d'exemples du chapitre, il existe un fichier d'inventaire nommé test-variable.inv qui permet de tester ces différents cas. Un bon moyen de tester cette hiérarchie entre les variables est de faire appel à la commande ansible avec le module debug (-m debug) et en paramètre les variables à vérifier (-a var=var1,var2) :

```
$ ansible -i test-variable.inv -m debug -a var=var1,var2 all
```

Cette commande vous donnera le résultat suivant :

```
localhost1 | SUCCESS => {
    "changed": false,
    "var1,var2": "(u'group1', u'group1')"
}
localhost2 | SUCCESS => {
    "changed": false,
    "var1,var2": "(u'group2', u'host_vars/localhost2')"
}
localhost3 | SUCCESS => {
    "changed": false,
    "var1,var2": "(u'group-123', u'group_vars/group3')"
}
localhost0 | SUCCESS => {
    "changed": false,
    "var1,var2": "(u'all', u'all')"
}
```

Les variables déclarées au niveau des machines prennent le pas sur celles au niveau du groupe et les fichiers host_vars et group_vars sont prioritaires sur les variables se trouvant dans le fichier d'inventaire.

Au niveau de l'utilisation des variables, tenez-vous-en à un type de déclaration et n'essayez pas de trop combiner entre elles toutes les options.

En règle générale, déclarez les variables au niveau des groupes dans le répertoire group_vars. Exceptionnellement, vous pouvez le faire au niveau du répertoire host_vars dans le cas d'une machine sur laquelle vous ne pouvez pas faire autrement.

Le fichier host, quant à lui, ne doit contenir que des déclarations de machines.

6. Gestion des différents inventaires

Vous avez vu le fonctionnement d'un inventaire simple.

Vous allez maintenant aborder une méthode pour mettre en commun un maximum d'éléments. Vous avez sûrement d'autres environnements que celui sur lequel vous faites vos tests (production, test d'intégration, etc.). Cela vous permettra de réutiliser le contenu de vos fichiers group_vars et host_vars.

Ce nouvel inventaire est constitué de deux serveurs Apache, prod-apache-1 et prod-apache-2, d'un serveur de répartition de charge, prod-haproxy dans le groupe haproxy, et de deux serveurs MySQL, prod-mysql-1 et prod-mysql-2.

Ci-dessous, le fichier d'inventaire à sauvegarder sous le nom production.inv :

```
[haproxy]
prod-haproxy

[apache]
prod-apache-1
prod-apache-2

[mysql]
prod-mysql-[1:2]
```

Le même inventaire au format YAML prendra la forme suivante :

```yaml
all:
  vars:
    ansible_connection: local
  children:
    haproxy:
      hosts:
        prod-haproxy: {}
    apache:
      hosts:
        prod-apache-1: {}
        Prod-apache-2: {}
    mysql:
      hosts:
        prod-mysql-[1:2]: {}
```

Le même inventaire au format YAML prendra la forme suivante :

```yaml
all:
  vars:
    ansible_connection: local
  children:
    haproxy:
      hosts:
        prod-haproxy: {}
    apache:
      hosts:
        prod-apache-1: {}
        Prod-apache-2: {}
    mysql:
      hosts:
        prod-mysql-[1:2]: {}
```

Sauvegardez ce fichier sous le nom de production.yml.

Maintenant, contactez les machines Apache de ce nouvel inventaire :

```
$ ansible -i production.yml -m ping apache
```

Ci-dessous le résultat de cette commande :

```
prod-apache-1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
prod-apache-2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Jusqu'ici, rien de particulier.

Combinez maintenant les deux inventaires en spécifiant le répertoire où ils se trouvent (-i .) :

```
$ ansible -i . -m ping apache
```

Ci-dessous le résultat de cette commande :

```
prod-apache-1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
rec-apache-1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
prod-apache-2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Ansible a réalisé automatiquement une fusion entre les deux inventaires ce qui va vous permettre de lancer des commandes sur toutes les machines de vos environnements.

Autre avantage de cette solution, la récupération des valeurs déjà définies dans le répertoire group_vars.

Si jamais vous deviez personnaliser des variables en fonction de votre environnement, vous pouvez à ce moment-là passer par une déclaration groupe:vars dans votre fichier d'inventaire (au format INI) ou en utilisant le champ vars (inventaire YAML). Attention toutefois à la hiérarchie de vos variables.

Il est possible de mélanger les inventaires au format INI et YAML. Cependant, ne faites pas de mélange ou faites-le de manière temporaire (lors de migration par exemple).

7. Création de groupes temporaires

Le mécanisme d'inventaire est très riche, mais tout n'est pas possible. Prenez un exemple tout simple : vous devez mettre à jour la version de SSL sur les machines CentOS 7.x.

Le premier réflexe est de créer un groupe et d'y ajouter les machines CentOS.

Or, le lendemain, vous avez à faire la même chose sur vos serveurs Debian en version 8 puis le jour suivant, avec un autre type de machine.

Au travers de cet exemple, vous vous rendez bien compte qu'il n'est pas toujours possible de tout prévoir.

La bonne nouvelle est qu'un mécanisme existe dans Ansible qui vous permet de créer des groupes temporaires et d'y affecter des machines en fonction de leurs caractéristiques (type de distribution, version du noyau, etc.). Cette affectation se fait en suivant la procédure ci-dessous :

- Lancement du setup sur l'ensemble des machines de l'inventaire.

- Ajout de chaque machine dans le groupe en fonction des caractéristiques des machines (version du kernel, famille de distribution, type d'OS, etc.).

- Lancement de tâches spécifiques au(x) groupe(s) de machines que vous venez de créer.

Appliquez maintenant ce principe à la mise à jour de la bibliothèque SSL des machines CentOS 7.x.

Dans un premier temps, le playbook récupère les facts de toutes vos machines sans distinction (option gather_facts: yes). Cette manipulation est suivie d'une instruction group_by avec la clé de regroupement (option key).

Ici, un exemple de regroupement en fonction du type de système et de sa version majeure :

```
- name: "Gather hosts facts"
 hosts: all
 gather_facts: yes
 gather_subset: "!all"
 tasks:
   - name: "Hosts affectation"
     group_by:
       key: "{{ansible_distribution}}-{{ansible_distribution_major_version}}"
   - debug:
     var: "ansible_distribution"
   - debug:
     var: "ansible_distribution_major_version"
```

L'instruction group_by va ajouter les différentes machines dans un nouveau groupe contenant le nom de la distribution (ex. : Debian, CentOS) ainsi que la version majeure (ex. : 6, 7 pour une CentOS ou 15, 16 pour une Ubuntu par exemple).

Maintenant que ces groupes sont disponibles, vous pouvez lancer les instructions en fonction du type de systèmes :

- yum pour les machines CentOS 7 ;

- apt pour les machines sous Debian 8.

Les deux modules utiliseront les mêmes options :

- le nom du package à mettre à jour (name=openssl) ;
- l'état attendu (state=latest pour indiquer de prendre la dernière version du package disponible).

Ci-dessous l'instruction pour les machines CentOS :

```
- name: "Update SSL on CentOS 7 machines"
  hosts: "CentOS-7"
  gather_facts: no
  tasks:
    - yum:
        name: "openssl"
        state: "latest"
```

Et maintenant les instructions pour les Debian :

```
- name: "Update SSL on debian 8 machines"
 hosts: "debian-8"
 gather_facts: no
 tasks:
   - apt:
       name: "openssl"
       state: "latest"
```

Chapitre 5

Sécuriser Ansible

SOLYTI

Chiffrement de fichiers

1. Contexte

Dans le chapitre de la découverte de l'inventaire, vous avez vu comment gérer ce dernier et stocker des identifiants ou des mots de passe. Petit problème, ces données sont stockées dans des fichiers plats. Quiconque ayant accès à vos inventaires pourrait en extraire des informations et s'en servir.

Dans ce qui va suivre, vous verrez un mécanisme qui permet de chiffrer vos fichiers afin d'éviter que des mots de passe en clair soient directement accessibles.

2. Stockage d'identifiants de connexion

La première étape va être de créer un fichier dans lequel vous allez stocker des informations de connexion à une base de données MySQL.

Vous utiliserez pour cela deux champs : intranet_mysql_db_user et intranet_mysql_db_password.

Ces deux identifiants vous serviront pour une hypothétique application intranet.

Ci-dessous, la déclaration que vous utiliserez comme exemple pour la suite :

```
intranet_mysql_db_user: "intranet"
intranet_mysql_db_password: "1ntran3t!"
```

3. Chiffrement du fichier entier

Le chiffrement d'éléments dans Ansible se fait à l'aide de l'outil ansible-vault. Ce dernier peut fonctionner avec un mot de passe ou en passant des fichiers en argument.

Dans les deux cas, pour le chiffrement d'un fichier entier, il faut lancer ansible-vault avec l'option encrypt (chiffrer en anglais) suivie du nom de fichier.

a. Par mot de passe

Ci-dessous la commande à lancer pour chiffrer le fichier intranet-pass.yml :

```
$ ansible-vault encrypt intranet-pass.yml
```

Ansible vous demandera alors un mot de passe qu'il faudra ensuite entrer une seconde fois pour confirmation. Ci-dessous la sortie de la commande lorsque le chiffrement s'est bien passé :

```
New Vault password:
Confirm New Vault password:
Encryption successful
```

Ci-dessous, un extrait du fichier intranet-pass.yml après chiffrement :

```
$ANSIBLE_VAULT;1.1;AES256
6536346161663[...]
```

L'utilisation du fichier se fera comme d'habitude avec l'option -e @fichier-vault.yml. Pour indiquer à Ansible qu'il faut déchiffrer les fichiers vault, vous ajouterez l'option --ask-vault-pass.

Ci-dessous le chargement du fichier intranet-pass.yml combiné à un appel à debug pour afficher le contenu de la variable intranet_mysql_db_user :

```
$ ansible -m debug -a var=intranet_mysql_db_user localhost \
          -e @intranet-pass.yml --ask-vault-pass
```

Ci-dessous le chargement du fichier intranet-pass.yml combiné à un appel à debug pour afficher le contenu de la variable intranet_mysql_db_user :

```
$ ansible -m debug -a var=intranet_mysql_db_user localhost \
          -e @intranet-pass.yml --ask-vault-pass
```

Ci-dessous le résultat de cet appel :

```
Vault password:
[WARNING]: provided hosts list is empty, only localhost is avail

localhost | SUCCESS => {
    "intranet_mysql_db_user": "intranet"
}
```

Ansible vous fait saisir le mot de passe et procède ensuite à l'exécution du module debug.

b. Utilisation d'un fichier

L'utilisation d'un mot de passe de chiffrement entraîne la saisie de ce dernier à chaque lancement d'Ansible. En effet, il n'est pas possible d'indiquer ce dernier par une variable d'environnement. Heureusement, vous pouvez utiliser un fichier : il suffit pour cela de mettre le mot de passe seul dans un fichier.

Reste à préciser l'emplacement à Ansible à l'aide des options suivantes :

- L'option --vault-password-file suivie du nom du fichier contenant le mot de passe.

- En exportant la variable DEFAULT_VAULT_PASSWORD_FILE avec l'emplacement du fichier.

Stockez le mot de passe « ansible » dans le fichier vault.key avec la commande suivante :

```
$ echo ansible > vault.key
```

Ci-dessous un exemple de déclaration de variable spécifiant l'emplacement du fichier contenant le mot de passe :

```
$ export DEFAULT_VAULT_PASSWORD_FILE=vault.key
```

Vous pouvez maintenant chiffrer le fichier intranet-pass.yml avec la commande suivante :

```
$ ansible-vault encrypt intranet-pass.yml \
  --vault-password-file vault.key
```

Vous devriez obtenir le message suivant : `Encryption successful`

Refaites le test précédent en ajoutant l'option --vault-password-file suivie du fichier contenant votre mot de passe :

```
$ ansible -m debug -a var=intranet_mysql_db_user localhost \
        -e @intranet-pass.yml --ask-vault-pass
```

Ci-dessous le résultat de votre commande :

```
localhost | SUCCESS => {
    "intranet_mysql_db_user": "intranet"
}
```

c. Déchiffrement d'un fichier

Vous souhaitez changer le contenu d'un fichier ? Vous avez à votre disposition plusieurs solutions avec ansible-vault :

- L'option decrypt qui va déchiffrer complètement le fichier pour vous permettre de modifier son contenu.

- L'option edit qui va éditer le fichier en fonction de la configuration de l'éditeur par défaut de votre système.

Ci-dessous, un exemple d'utilisation de l'option edit avec ansible-vault :

```
$ ansible-vault edit intranet-pass.yml \
    --vault-password-file ./vault.key
```

Reste ensuite à effectuer votre modification comme vous l'auriez fait pour n'importe quel autre fichier.

d. Changement du mot de passe de chiffrement

Votre mot de passe ou le fichier de chiffrement a été compromis et vous souhaitez refaire votre chiffrement. Dans ce cas, il faudra passer l'option rekey à ansible-vault.

Ci-dessous, la commande à lancer :

```
$ ansible-vault rekey intranet-pass.yml
```

Il vous sera alors demandé l'ancien mot de passe suivi du nouveau comme indiqué ci-dessous :

```
Vault password:
New Vault password:
Confirm New Vault password:
Rekey successful
```

Dans le cas où vous feriez appel à des fichiers de mots de passe, vous serez intéressé d'apprendre l'existence des options --new-vault-password-file et --vault-password-file.

Ci-dessous, un exemple de rechiffrement d'un fichier en passant par l'ancien fichier vault.key et le nouveau fichier new-vault.key :

```
$ ansible-vault rekey \
   --new-vault-password-file ./new-vault.key \
   --vault-password-file ./vault.key \
 intranet-pass.yml
```

Et ci-dessous le résultat en cas de succès :

```
Rekey successful
```

4. Chiffrement d'un champ

Le chiffrement d'un fichier est un moyen très efficace pour protéger vos identifiants et mots de passe dans un fichier YAML. En revanche, le fichier perd complètement les informations sur les variables qui s'y trouvent. Une solution intermédiaire (disponible depuis la version 2.3) est de passer par le chiffrement des champs d'un fichier. Vous gardez ainsi le nom de vos variables, mais leur contenu reste inconnu.

Pour cela ansible-vault offre l'option encrypt_string suivie de la chaîne à chiffrer.

Ci-dessous un exemple de lancement pour la chaîne 1ntran3t! :

```
$ ansible-vault encrypt_string '1ntran3t!' \
    --vault-password-file ./vault.key
```

Ci-dessous, le résultat de cette commande :

```
!vault |
          $ANSIBLE_VAULT;1.1;AES256
          64663162646664[...]
          65633133396562[...]
          63323332396364[...]
          36383461366234[...]
          3934
Encryption successful
```

Reste maintenant à copier-coller ce résultat dans le fichier YAML partial-vault.yml. Ci-dessous le contenu que vous utiliserez dans votre exemple :

```
intranet_mysql_db_user: "intranet"
intranet_mysql_db_password: !vault |
          $ANSIBLE_VAULT;1.1;AES256
          64663162646664[...]
          65633133396562[...]
          63323332396364[...]
          36383461366234[...]
          3934
```

Lancez maintenant le test d'affichage de la variable intranet_mysql_db_password avec en paramètre le fichier partial-vault.yml. En reprenant les précédents lancements, la commande devrait ressembler à ce qui suit :

```
$ ansible -m debug -a var=intranet_mysql_db_password localhost \
          -e @partial-vault.yml --vault-password-file ./vault.key
```

Ci-dessous, vous retrouvez le résultat de votre commande :

```
localhost | SUCCESS => {
    "intranet_mysql_db_password": "1ntran3t!"
}
```

Chapitre 6

Playbooks et Templates

Le moteur de template Jinja : principe de fonctionnement

Dans les exemples qui vont suivre, vous allez voir comment utiliser Ansible pour récupérer les informations qui caractérisent les machines afin de les répertorier. Vous aurez besoin pour cela d'aborder le fonctionnement du moteur Jinja.

Jinja est un moteur de template écrit en Python. Il est inspiré du moteur Django tout en restant plus simple dans son utilisation. Le principe est le suivant : un canevas va contenir toute la mise en page. Jinja charge ce patron et réalise la substitution des champs en fonction du contexte d'exécution.

On utilise généralement ce genre d'outil pour afficher une information dans une page web en fonction de variables (information sur un utilisateur, affichage de la date du jour, etc.). Pour bien comprendre comment fonctionne ce type de moteur, prenez le cas d'une page HTML présentant le nom d'une machine :

```
<html>
 <head>
   <title>Machine rec-apache-1</title>
 </head>
 <body>
   <p>Cette machine s'appelle rec-apache-1</p>
 </body>
</html>
```

Vous y retrouvez le nom de la machine à deux endroits distincts : dans la balise titre de la page (balise <title>...</title>) et dans le paragraphe (balise ...) du corps de la page (balise <body>...</body>).

Imaginez maintenant que le nom de la machine soit stocké dans la variable inventory_hostname. Pour rendre la page précédente générique, il vous suffit de substituer le nom de la machine (rec-apache-1) par la variable entourée d'accolades ({{ et }}). Dans ce cas, cette page pourrait donc ressembler au patron suivant :

```
<html>
 <head>
   <title>Machine {{inventory_hostname}}</title>
 </head>
 <body>
   <p>Cette machine s'appelle {{inventory_hostname}}</p>
 </body>
</html>
```

SOLYTI

Par la suite, vous stockez ce patron dans le fichier machine.html.j2.

Vous n'êtes pas obligé d'utiliser l'extension .j2, mais c'est une bonne pratique. En effet, elle vous permettra de distinguer rapidement un fichier simple (test.html) d'un fichier Jinja (test.html.j2). Une autre bonne pratique est de garder l'extension du type de fichier original suivie de .j2. Vous saurez ainsi rapidement quel est le type de fichier avec lequel vous allez travailler (.sh.j2 pour un template de shell, .sql.j2 pour un template de fichier SQL, etc.).

En passant la variable inventory_hostname et le patron à Jinja, ce dernier se charge alors de faire la substitution à votre place.

Passons maintenant aux travaux pratiques : Ansible est construit sur le moteur Jinja et le nom de la machine courante est stocké dans la variable inventory_hostname. Ansible dispose d'un module spécialement conçu pour ce besoin : le module template.

Ce module va avoir besoin des options suivantes :

- l'emplacement du fichier source (option src) ;
- l'emplacement où vous voulez faire le rendu (option dest).

Comme d'habitude, vous allez devoir donner un inventaire à Ansible (option -i suivie d'un fichier d'inventaire) ainsi que la liste des machines sur lesquelles vous allez travailler (dans ce cas le groupe all).

Autre astuce, vous allez lancer le rendu des fichiers HTML seulement en local.

Pour cela, utilisez l'option -c local pour préciser que tout se fera depuis la machine qui lance Ansible.

En résumant toutes les différentes options, vous devriez être en mesure de lancer le test avec la commande suivante :

```
$ ansible -i inventaire-apache.yml -m template \
 -a "src=machine.html.j2 \
     dest=$PWD/{{inventory_hostname}}.html" \
 -c local all
```

Ci-dessous une sortie partielle du résultat de cette commande :

```
rec-apache-2 | SUCCESS => {
   "changed": true,
 [...]
}
rec-apache-1 | SUCCESS => {
   "changed": true,
[...]
}
```

Afin de simplifier un peu le lancement, écrivons le playbook équivalent à ce vous venez de lancer en ligne de commande. Avant d'y arriver, vous allez devoir faire quelques adaptations.

Tout d'abord, Ansible n'accepte pas directement les variables d'environnement. Dans le cas de PWD, cette variable peut être remplacée par playbook_dir. Cette dernière contient en effet le chemin absolu du répertoire contenant le playbook.

Autre point, la gestion de la connexion peut se faire avec l'option -c. Néanmoins, il vaut mieux passer par le champ connection (en lui passant la valeur local) au niveau du playbook.

En combinant tout ceci, vous obtenez le playbook suivant :

```
- name: "Generate html file for each host"
 hosts: all
 connection: local
 tasks:
   - name: "html file generation"
     template:
       src: "machine.html.j2"
       dest: "{{playbook_dir}}/{{inventory_hostname}}.html"
```

Sauvegardez ce playbook sous le nom d'inventory.yml et passez-le à ansible-playbook en y ajoutant le nom de l'inventaire (-i inventaire-apache.yml). En combinant toutes ces options, vous obtenez la commande suivante :

```
$ ansible-playbook -i inventaire-apache.yml inventory.yml
```

Ci-dessous le résultat de cette commande :

```
PLAY [Generate html file for each host] *************************

TASK [Gathering Facts] ******************************************
ok: [rec-apache-1]
ok: [rec-apache-2]

TASK [html file generation] *************************************
ok: [rec-apache-1]
ok: [rec-apache-2]

PLAY RECAP ******************************************************************
rec-apache-1     : ok=2    changed=0    unreachable=0    failed=0
rec-apache-2     : ok=2    changed=0    unreachable=0    failed=0
```

Si les fichiers HTML sont présents avec le bon contenu, il n'y aura pas de mise à jour (ligne verte commençant par OK). Ceci est tout à fait normal puisque les fichiers ont déjà été générés par la commande ansible. C'est un point important dans la gestion de la réentrance pour savoir si des modifications ont eu lieu dans un système entre deux exécutions d'Ansible.

Suite à ce lancement, vous devriez trouver un fichier HTML par machine dans le répertoire courant :

```
$ ls *.html
rec-apache-1.html  rec-apache-2.html
```

Jetez un coup d'œil sur le contenu des fichiers pour voir s'ils sont bien différents en fonction du nom des machines de l'inventaire.

Pour cela, vous pouvez utiliser la commande grep avec les options suivantes :

- le motif à chercher dans le fichier (title par exemple) ;
- la liste des fichiers générés.

Ci-dessous le résultat d'une commande qui vous permet de réaliser cette comparaison :

```
$ grep title rec-apache-*.html
rec-apache-1.html:    <title>Machine rec-apache-1</title>
rec-apache-2.html:    <title>Machine rec-apache-2</title>
```

Template Jinja

Vous avez vu un mécanisme simple qui permet de générer des fichiers sur la machine locale en partant des informations de l'inventaire. Vous allez maintenant voir comment utiliser les informations remontées par le module setup (champ gather_facts des playbooks). Vous vous servirez ensuite de ces variables dans la génération du template.

Vous allez pour cela aborder plusieurs points :

- Savoir réaliser une boucle avec un template Jinja.

- Trouver la variable Ansible remontée par le module setup contenant ce qui vous intéresse.

- Comprendre comment lancer les modules aux bons endroits avec l'option connection.

1. Mécanisme de boucle sur tableau avec Jinja

Sous Jinja, une boucle se déclare avec les éléments suivants :

- le mot-clé for ;
- le nom de la variable d'itération ;
- le mot-clé in ;
- le tableau sur lequel vous allez itérer.

Ci-dessous un exemple de ce type de boucle :

```
{% for element in liste_element %}
Le contenu de la variable element est le suivant : {{element}}
{% endfor %}
```

Ici la variable liste_element sera un tableau d'éléments. Chaque élément générera une ligne.

Comme vous l'avez vu plus tôt, en plus des tableaux, vous pouvez également utiliser des tableaux de hachage. Dans ce cas, il faudra :

- faire appel à la fonction items du tableau de hachage ;
- remplacer la variable d'itération simple par un couple de variables clé/valeur.

Ci-dessous un exemple simple de ce type de déclaration :

```
{% for cle, valeur in cles_valeurs.items() %}
La clé contient {{cle}} tandis que la valeur contient {{valeur}}
{% endfor %}
```

2. À la recherche de la bonne variable Ansible

Maintenant que vous avez vu comment itérer sur un tableau contenant plusieurs éléments, vous allez pouvoir afficher dans le template les interfaces réseau des différentes machines.

Première chose à faire : trouvez l'information nécessaire dans les variables du module setup d'Ansible.

Le plus simple pour cela est de lancer ansible en lui passant :

- le module setup (-m setup) ;

- l'inventaire (-i inventaire-apache.yml) ;

- la machine sur laquelle vous allez travailler (rec-apache-1 par exemple).

En effet, rien ne sert de récupérer les éléments de toutes les machines. Il vaut mieux faire la recherche sur une seule machine. Cela limitera le nombre de variables renvoyées à l'utilisateur (surtout si vous travaillez sur plusieurs milliers de machines).

Pour trouver votre bonheur au niveau des variables, vous pouvez indifféremment lancer une commande grep sur ce que vous cherchez (valable dans le cas des variables simples) ou bien faire une redirection dans un fichier ou dans un pipe de la commande less.

Dans le cas présent, comme il s'agit de retrouver le tableau contenant la liste des interfaces réseau, vous allez rediriger ce résultat dans un fichier resultat.txt.

```
$ ansible -m setup -i inventaire-apache.yml \
                        rec-apache-1 > resultat.txt
```

Reste maintenant à chercher ce qui vous intéresse dans ce fichier à l'aide de votre éditeur préféré (vi, Emacs, Kate, gedit, etc.).

Ci-dessous, l'extrait qui devrait vous intéresser :

```
rec-apache-1 | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "10.0.2.15",
            "192.168.66.2"
        ],
[...]
```

Il s'agit de la variable ansible_all_ipv4_addresses.

Dans les interfaces réseau, on peut également noter les variables suivantes :

- ansible_all_ipv6_addresses : liste des interfaces réseau IPv6.

- ansible_default_ipv4/6 : information concernant l'interface par défaut IPv4/6.

- ansible_interfaces : liste des interfaces réseau de la machine (lo, enp0s3, eth0 etc.).

- ansible_eth0/enp0s3 : une variable par interface avec les informations s'y rattachant.

Dans les éléments qui peuvent également vous intéresser, vous retrouvez des informations sur :

- les disques/partitions de la machine (ansible_devices ou ansible_mounts) ;
- les caractéristiques de la machine (ansible_processor donne la liste des processeurs, ansible_memtotal_mb la quantité de mémoire) ;
- les informations sur l'utilisateur distant (ansible_user_id).

3. Affichage des interfaces réseau

Vous savez maintenant que les interfaces réseau d'une machine se trouvent dans la variable ansible_all_ipv4_addresses.

Vous allez donc pouvoir boucler dessus pour en afficher une adresse par ligne.

Ci-dessous le code du template que vous allez utiliser :

```
<html>
 <head>
   <title>Machine {{inventory_hostname}}</title>
 </head>
 <body>
   <p>Cette machine s'appelle {{inventory_hostname}}</p>
   <p>Ci-dessous la liste de ces adresses :</p>
   <ul>
{% for ip in ansible_all_ipv4_addresses %}
     <li>{{ip}}</li>
{% endfor %}
   </ul>
 </body>
</html>
```

SOLYTI

Relancez le playbook inventory.yml précédemment créé et consultez la liste des interfaces avec la commande grep li (ce qui retournera les lignes contenant les adresses IP) sur l'ensemble des fichiers HTML :

```
$ grep li *.html
```

Ci-dessous le résultat correspondant à cette commande :

```
rec-apache-1.html:    <p>Ci-dessous la liste de ces adresses :</p>
rec-apache-1.html:      <li>10.0.2.15</li>
rec-apache-1.html:      <li>192.168.66.180</li>
rec-apache-2.html:    <p>Ci-dessous la liste de ces adresses :</p>
rec-apache-2.html:      <li>10.0.2.15</li>
rec-apache-2.html:      <li>192.168.66.180</li>
```

Ce résultat est suspect puisque les adresses sont les mêmes sur les deux machines. En creusant un peu, on se rend compte qu'en réalité, Ansible vous a renvoyé les adresses IP de la machine qui a lancé l'interpréteur.

SOLYTI

Réduction des opérations impactantes

De manière générale, il faut réduire au maximum les opérations inutiles. Parmi ces opérations, les arrêts/relances sont une source d'indisponibilité de service ou de perte de temps. Dans le cas présent, il n'y a pas de grosses conséquences à faire cet arrêt/relance. En revanche, ce point peut devenir gênant dans le cas d'une application à disposition d'utilisateurs.

Dans la suite, vous aborderez deux techniques permettant de réduire la fréquence de ces opérations :

- via l'utilisation de tags ;

- par l'utilisation de variables et le conditionnement du lancement d'opérations ;

- par un mécanisme de handler.

1. Mécanisme des tags

Une première façon d'exclure une tâche d'un flot d'exécutions est de passer par les tags. Les tags peuvent se définir à plusieurs niveaux :

- sur le playbook en lui-même ;

- au niveau d'une tâche.

a. Déclaration d'un tag

L'ajout d'une annotation se fait avec le champ tag. Si vous ajoutez un tag restart sur la tâche de redémarrage du service Apache, le code de la tâche prendra la forme suivante :

```
- name: "Start apache service"
 service:
   name: "httpd"
   state: "restarted"
   enabled: yes
 tags: [ "restart" ]
```

b. Comment lister les tags d'un playbook ?

Pour obtenir la liste des tags à disposition sur un playbook, en plus de la commande ansible-playbook suivie du nom du playbook, ajoutez l'option --list-tags.

Pour le playbook install-apache.yml, la commande sera la suivante :

```
$ ansible-playbook install-apache.yml --list-tags
```

Ci-dessous le résultat de cette commande :

```
playbook: install-apache.yml

 play #1 (inventaire): Apache installation     TAGS: []
     TASK TAGS: [restart]
```

Le tag restart est bien présent sur le playbook d'installation d'Apache.

c. Sélection ou exclusion d'un tag

Afin d'exclure ou sélectionner un tag, vous avez à votre disposition deux options :

--tags LIST_TAGS : sélection des tâches à exécuter ;

--skip LIST_TAGS : exclusion de tâches.

Ci-dessous la commande permettant de ne lancer que le redémarrage d'Apache :

```
$ ansible-playbook -i inventaire-apache.yml install-apache.yml \
                   --tags restart
```

Ci-dessous le résultat de cette commande :

```
PLAY [Apache installation] ***********************************

TASK [Gathering Facts] ***************************************
ok: [central-inventory]

TASK [Start apache service] **********************************
changed: [central-inventory]

PLAY RECAP ***************************************************
central-inventory     : ok=2    changed=1    unreachable=0    failed=0
```

Ci-dessous la même commande, mais en excluant le tag restart :

```
$ ansible-playbook -i inventaire-apache.yml install-apache.yml \
                   --skip restart
```

Ci-dessous le résultat de cette commande :

```
PLAY [Apache installation] *********************************

TASK [Gathering Facts] *************************************
ok: [central-inventory]

TASK [Apache package installation] *****************************
ok: [central-inventory]

TASK [Apache configuration] ********************************
ok: [central-inventory]

PLAY RECAP *************************************************
central-inventory     : ok=3   changed=0  unreachable=0  failed=0
```

d. Gather facts et le tag always

À y regarder de plus près, la tâche de récupération des informations sur la machine a été lancée systématiquement (Gathering Facts), quelle que soit la valeur des options --tags ou --skip. C'est en raison du fait que son exécution est conditionnée par l'option gather_facts.

Dans le cas où vous voudriez rendre l'exécution d'une tâche obligatoire, vous pouvez passer par l'utilisation d'un tag particulier : always. Dans ce cas, la tâche sera toujours exécutée.

2. Utilisation d'une variable sur une tâche

Vous avez vu le fonctionnement des variables Ansible au niveau de l'inventaire. Il est également possible de stocker le résultat des exécutions d'une tâche pour pouvoir s'en servir plus tard.

Reprenez le playbook et appliquez-y les modifications suivantes :

- stockage de la tâche de configuration avec le mot-clé register suivi d'un nom de variable (exemple : apache_conf) ;
- conditionnement d'un redémarrage (module service avec l'option state: restarted) sur le résultat de la tâche de configuration.

Le conditionnement se fera avec le mot-clé when suivi de la condition à tester.

Afin de savoir quoi utiliser dans la variable, vous allez temporairement ajouter une instruction debug avec l'option var: apache_conf. Ceci permettra de scruter le contenu de la variable et de savoir sur quoi se baser pour faire l'arrêt/relance.

Ci-dessous le playbook correspondant au test à réaliser :

```
- name: "Apache installation"
 hosts: inventory
 tasks:
   - name: "Apache configuration"
     template:
       src: "inventory.conf.j2"
       dest: "/etc/httpd/conf.d/inventory.conf"
       owner: "apache"
       group: "apache"
     # register execution result for template creation
     register: apache_conf
   # Instruction permettant de scruter le contenu de la variable
   - debug: var=apache_conf
```

Lancez le playbook et concentrez-vous sur le contenu de l'instruction debug :

```
TASK [debug] ********************************************************
ok: [central-inventory] => {
    "changed": false,
    "apache_conf": {
        "changed": false,
        "diff": {
            "after": {
                "path": "/etc/httpd/conf.d/inventory.conf"
            },
            "before": {
                "path": "/etc/httpd/conf.d/inventory.conf"
            }
        },
        "gid": 48,
        "group": "apache",
        "mode": "0644",
        "owner": "apache",
        "path": "/etc/httpd/conf.d/inventory.conf",
        "size": 176,
        "state": "file",
        "uid": 48
    }
}
```

Cette instruction va vous permettre de faire un petit tour des champs à utiliser pour l'instruction when.

Certains champs comme changed sont toujours présents. Ce dernier détermine s'il y a eu un changement. D'autres comme diff ne seront pas forcément présents. Attention donc à bien vérifier leur présence avant de les utiliser.

Il existe d'autres champs très standards. C'est le cas des modules gérant un fichier (file, lineinfile, copy, template), dans lesquels on retrouvera toujours les champs suivants :

- owner ou group : le propriétaire/groupe du fichier ;

- mode : droits sur le fichier (au format Unix) ;

- path : le chemin du fichier.

Les modules de commande (shell ou command) ont également des champs toujours présents :

- rc : code retour récupéré par le système au lancement du script ;
- stdout : sortie standard brute ;
- stderr : idem sortie d'erreur ;
- stdout_lines : même chose que stdout, mais sous la forme d'un tableau.

Dans le cas présent, vous devez déterminer s'il faut oui ou non redémarrer le serveur Apache.

Enregistrez le résultat de la tâche de mise à jour de la configuration.

Utilisez la condition is changed sur la variable apache_conf afin de conditionner l'arrêt/relance avec une instruction when.

Ci-dessous la nouvelle version du playbook :

```
- name: "Apache installation"
  hosts: inventory
  tasks:
    - name: "Apache package installation"
      yum:
        name: "httpd"
        state: "present"
    - name: "Apache configuration"
      template:
        src: "inventory.conf.j2"
        dest: "/etc/httpd/conf.d/inventory.conf"
        owner: "apache"
        group: "apache"
      # register execution result for template creation
      register: apache_conf
    # debug instruction to get apache_conf content
    - debug: var=apache_conf
```

```
    - name: "Allow http connections"
      firewalld:
        service: "http"
        permanent: yes
        state: "enabled"
    - name: "Restart apache service"
      service:
        name: "httpd"
        state: "restarted"
      # restart if necessary
      when: apache_conf is changed
    - name: "Start apache service"
      service:
        name: "httpd"
        state: "started"
        enabled: yes
```

Vous pouvez récupérer ce playbook dans le fichier install-apache-restart-by-register.yml dans le répertoire des fichiers d'exercice de ce chapitre. En reprenant les options précédemment utilisées, la commande sera la suivante :

```
$ ansible-playbook -i inventaire-apache.yml \
    install-apache-restart-by-register.yml
```

Ci-dessous le résultat de l'exécution :

```
PLAY [Apache installation] *********************************

TASK [Gathering Facts] *************************************
ok: [central-inventory]

TASK [Apache package installation] *****************************
ok: [central-inventory]

TASK [Apache configuration] ********************************
ok: [central-inventory]

TASK [debug] ***********************************************
ok: [central-inventory] => {
   "changed": false,
   "apache_conf": { [...] }
}

TASK [Restart apache service] ******************************

TASK [Allow http connections] ******************************
ok: [central-inventory]

TASK [Start apache service] ********************************
ok: [central-inventory]

PLAY RECAP *************************************************
central-inventory  : ok=5   changed=0   unreachable=0   failed=0
```

Ici, la tâche de relance du service Apache n'est pas exécutée. Vous disposez maintenant d'un playbook réentrant.

3. Utilisation d'un handler

a. Pourquoi utiliser un handler ?

Vous l'avez vu, le playbook est réentrant. Néanmoins, vous avez dû mettre en œuvre pas mal de choses pour y arriver :

- déclaration de variable ;
- récupération du contenu d'une variable ;
- ajout d'une condition d'exécution sur une opération.

Sans compter que, si vous vouliez reprendre l'exécution du playbook à partir de l'instruction Restart apache service (en utilisant l'option --start-at-task="Restart apache service"), vous obtiendriez l'erreur suivante :

```
The conditional check 'apache_conf is changed' failed. The error was:
error while evaluating conditional (apache_conf is changed):
'apache_conf' is undefined
```

En effet, comme vous n'avez pas lancé l'instruction Apache configuration, la variable apache_conf est inconnue.

Un autre cas pourrait survenir : l'utilisation de tags. En effet, les tags sont là pour permettre de sélectionner certaines tâches à l'exécution d'un playbook.

Pour toutes ces raisons afin de simplifier, vous allez aborder un autre mécanisme : les handlers.

b. Déclaration d'un handler

Les handlers sont des tâches un peu particulières puisqu'elles ne sont appelées qu'en cas de besoin. Leur utilisation se fera de la manière suivante :

- Déclaration dans la section handlers avec un nom explicite.
- Ajout d'un champ notify avec le nom du handler à appeler au niveau de la tâche.

Dans le précédent playbook, vous allez donc réaliser les actions suivantes :

- Déplacement de la tâche de redémarrage dans une section handlers.

- Suppression de la condition d'exécution sur le handler (suppression du champ when).

- Suppression du champ register sur la tâche de configuration Apache.

- Ajout du champ notify sur cette même tâche avec le nom du handler (notify: "Restart apache service").

Ci-dessous le playbook correspondant à ces modifications :

```yaml
- name: "Apache installation"
  hosts: inventory
  handlers:
    - name: "Restart apache service"
      service:
        name: "httpd"
        state: "restarted"
  tasks:
    - name: "Apache package installation"
      yum:
        name: "httpd"
        state: "present"
    - name: "Apache configuration"
      template:
        src: "inventory.conf.j2"
        dest: "/etc/httpd/conf.d/inventory.conf"
        owner: "apache"
        group: "apache"
      # Notify Apache restart handler
      notify: [ "Restart apache service" ]
```

```yaml
- name: "Allow http connections"
  firewalld:
    service: "http"
    permanent: yes
    state: "enabled"
- name: "Start apache service"
  service:
    name: "httpd"
    state: "started"
    enabled: yes
```

Outre le fait que le playbook est plus simple et qu'il supportera les lancements qui sautent certaines étapes, le handler vous permettra également de gérer très facilement le cas où plusieurs conditions de lancement d'une tâche existeraient.

c. Exemple de lancement

Ce fichier sera stocké dans le fichier install-apache-restart-by-handler.yml.

En reprenant les précédentes exécutions et en les adaptant, vous devriez obtenir le résultat suivant :

```
PLAY [Apache installation] ***********************************

TASK [Gathering Facts] ***************************************
ok: [central-inventory]

TASK [Apache package installation] ***************************
ok: [central-inventory]

TASK [Apache configuration] **********************************
ok: [central-inventory]

TASK [Allow http connections] ********************************
ok: [central-inventory]

TASK [Start apache service] **********************************
ok: [central-inventory]

PLAY RECAP ***************************************************
central-inventory     : ok=4    changed=0    unreachable=0  failed=0
```

Lorsqu'il n'y a pas de changement, vous ne voyez pas l'appel à la fonction de redémarrage.

Ci-dessous un exemple dans lequel la configuration d'Apache change :

```
PLAY [Apache installation] ***********************************

TASK [Gathering Facts] ***************************************
ok: [central-inventory]

TASK [Apache package installation] ***************************
ok: [central-inventory]

TASK [Apache configuration] **********************************
changed: [central-inventory]

TASK [Allow http connections] ********************************
ok: [central-inventory]

TASK [Start apache service] **********************************
ok: [central-inventory]

RUNNING HANDLER [Restart apache service] *********************
changed: [central-inventory]

PLAY RECAP ***************************************************
central-inventory : ok=5    changed=2    unreachable=0    failed=0
```

Chapitre 7

Les Roles Ansible

SOLYTI

Introduction aux rôles Ansible

Comme vu dans les chapitres précédents, les playbooks vous permettent d'enchaîner plusieurs actions (templates, shell, copie de fichiers) ainsi que l'endroit où vous devez les réaliser (méthode de connexion, délégation d'opération, etc.).

Seul problème : quand vous aurez écrit une dizaine de playbooks avec pour chacun plusieurs templates et la gestion de plusieurs fichiers, vous risquez d'avoir un peu de mal à vous organiser. Bien sûr, rien ne vous empêche de commencer à créer des sous-répertoires pour y placer vos templates classés par playbooks.

Viendra aussi inévitablement le besoin de réutiliser des instructions. L'instruction include peut vous aider, mais ce mécanisme ne fonctionnera pas dans tous les cas.

La notion de rôle répond à ces deux problématiques :

- Réutiliser du code dans plusieurs playbooks de manière modulaire.
- Organiser le code se trouvant dans les playbooks.

Dans le cas qui va suivre, vous travaillerez sur la réutilisation du playbook d'installation d'Apache nécessaire au fonctionnement de MediaWiki.

1. Structure d'un rôle

Un rôle est une structure arborescente contenant certains répertoires et fichiers. Les rôles présents dans le répertoire roles seront automatiquement utilisables par vos playbooks. Il est également possible de les mettre à disposition d'Ansible de la manière suivante :

- Modification de la variable d'environnement DEFAULT_ROLES_PATH.

- Modification de l'option roles_path du fichier de configuration Ansible.

Chaque rôle aura donc un répertoire du même nom dans le répertoire roles. Dans un rôle, vous pourrez utiliser les répertoires suivants :

- files : tous les fichiers à copier avec le module copy.
- templates : fichiers de template Jinja.
- tasks : liste des instructions à exécuter (dans le fichier main.yml).
- handlers : même chose pour les instructions handlers (fichier main.yml).

- vars : fichier contenant des déclarations de variables (fichier main.yml).
- defaults : valeurs par défaut (fichier main.yml).
- meta : dépendances du rôle (fichier main.yml).

Comme vous pouvez le constater, certains répertoires doivent obligatoirement contenir un fichier main.yml sinon ils ne sont pas pris en compte. Pour files et templates, il n'y a pas besoin de fichier main.yml puisqu'il s'agit juste d'un espace de stockage.

Si un répertoire n'est pas présent, il n'est pas pris en compte. Point important : aucun répertoire n'est obligatoire.

Ainsi, vous pouvez tout à fait créer un rôle qui ne contient que le fichier defaults/main.yml ou tasks/main.yml. Dans le premier cas, il chargera des valeurs par défaut, et dans le second cas il lancera les opérations contenues dans le fichier tasks/main.yml.

2. Votre premier rôle : installation d'Apache

Reprenez le playbook écrit pour l'installation d'Apache (playbook install-apache-restart-by-handler.yml). Vous découperez les instructions de ce playbook pour une installation générique du serveur Apache. Vous pourrez ainsi l'utiliser plus tard pour le wiki.

Appellez ce rôle d'installation : apache. La racine de ce rôle sera donc le répertoire roles/apache au même niveau que le playbook.

Pour reprendre les fonctionnalités du playbook, ce rôle devra disposer des éléments suivants :

- Un fichier tasks/main.yml pour les opérations à lancer.
- Un fichier handlers/main.yml pour le handler de redémarrage apache.

Du côté du fichier tasks/main.yml, reprenez la liste des actions qui se trouvent dans la section tasks en faisant attention de supprimer l'indentation ainsi que le mot-clé tasks.

Supprimez également la gestion de la configuration Apache (vous verrez comment l'utiliser dans le chapitre suivant). Le fichier devrait donc ressembler à ceci :

```
- name: "apache installation"
 yum:
   name: "httpd"
   state: "present"
- name: "apache service activation"
 service:
   name: "httpd"
   state: "started"
   enabled: yes
```

Dans le fichier handlers/main.yml mettez le contenu de la section handlers. De la même manière que pour tasks/main.yml, supprimez l'indentation ainsi que le mot-clé handlers. Pour résumer, le fichier devrait ressembler à ce qui suit :

```
- name: "apache restart"
 service:
   name: "httpd"
   state: "restarted"
```

Enfin, supprimez du playbook les sections tasks et handlers. Remplacez-les par une section roles.

Dans cette section, vous listerez les rôles à appeler (ici, apache). Le nom du rôle peut éventuellement être précédé du mot-clé role. Ce playbook portera le nom de install-inventaire.yml.

Ci-dessous le contenu de ce fichier :

```
- name: "Apache installation"
 hosts: inventory
 roles:
   - role: "apache"
```

En résumant toutes ces informations, la commande devrait être la suivante :

```
$ ansible-playbook -i inventaire-apache.yml install-inventaire.yml
```

SOLYTI

Ci-dessous le résultat de cette exécution :

```
PLAY [Installation apache] ***********************************

TASK [Gathering Facts] ***************************************
ok: [inventaire-central]

TASK [apache : Installation package apache] ******************
ok: [inventaire-central]

TASK [apache : Démarrage du service apache] ******************
ok: [inventaire-central]

PLAY RECAP ***************************************************
inventaire-central  : ok=4   changed=0   unreachable=0  failed=0
```

Rien de particulier à signaler sur cette exécution si ce n'est qu'Ansible préfixe le nom des tâches exécutées par leur contexte (ici, le rôle apache). Cette information peut être intéressante pour réaliser des diagnostics sur ce qui a été lancé.

Parfait, vous disposez d'un rôle générique qui permet d'installer Apache. Vous allez maintenant créer un nouveau rôle apache-inventory pour prendre en charge la partie spécifique. Pour cela, créez les éléments suivants :

- Un fichier tasks/main.yml avec l'instruction de dépose et option de notification.
- Le fichier templates/inventaire.conf.j2 avec la configuration qui vous intéresse.

Ci-dessous, le contenu du fichier tasks/main.yml :

```
- name: "Apache configuration"
  template:
    src: "inventaire.conf.j2"
    # apache_conf_d pointe vers la conf apache
    dest: "/etc/httpd/conf.d/inventaire.conf"
    owner: "apache"
    group: "apache"
  # En cas de changement, redémarrage d'apache
  notify: [ "apache restart" ]
```

Modifiez également le playbook install-inventaire.yml dans lequel vous ajouterez l'appel au rôle apache-inventory.

Ci-dessous le contenu du playbook correspondant à cette modification :

```
- name: "Apache installation for inventory"
 hosts: inventory
 roles:
   - role: "apache"
   - role: "apache-inventory"
```

Reprenez la même commande que précédemment.

Ci-dessous le résultat de l'exécution :

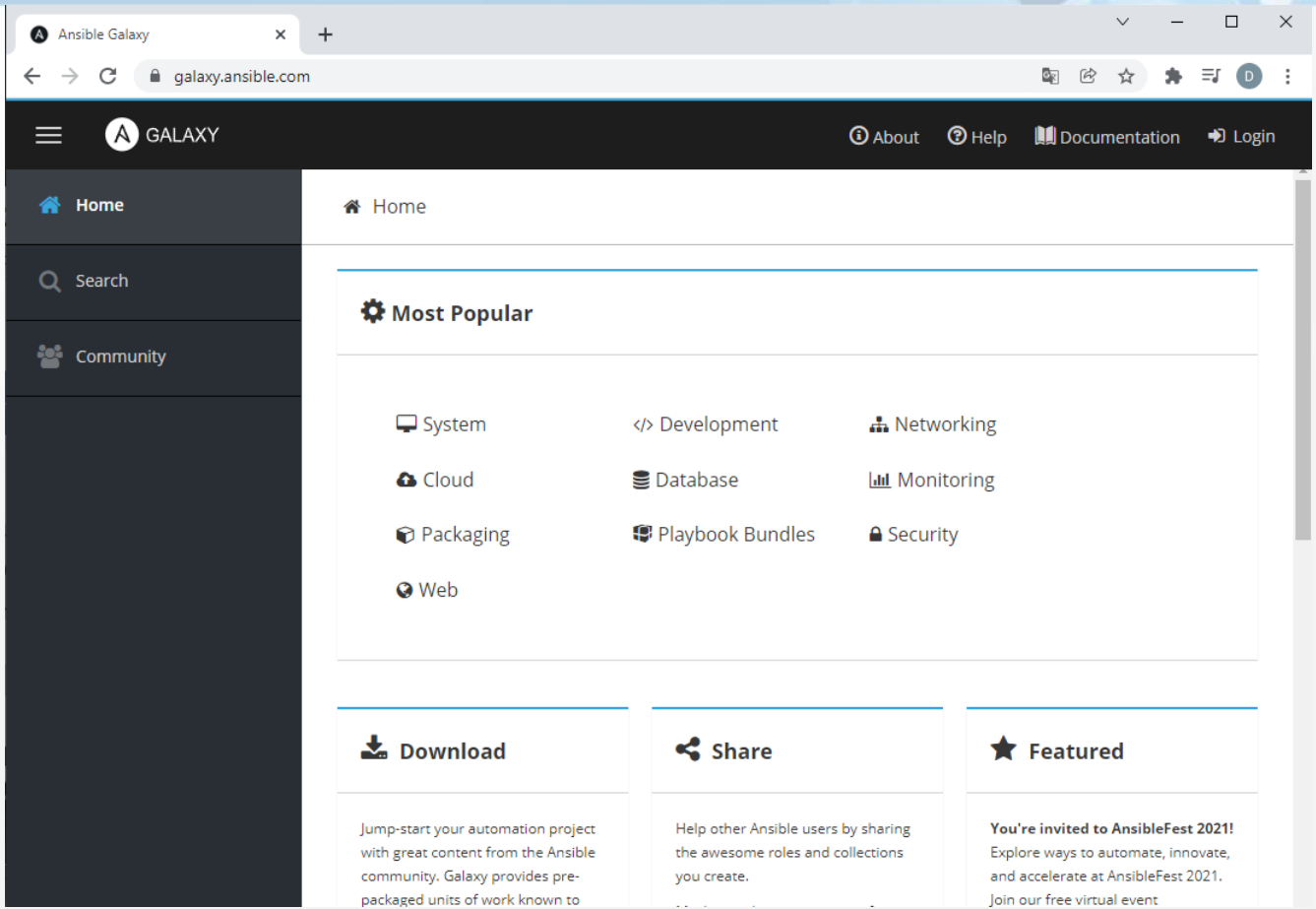
```
PLAY [Apache installation for inventory] ***********************

TASK [Gathering Facts] *****************************************
ok: [inventaire-central]

TASK [apache : apache installation] ****************************
ok: [inventaire-central]

TASK [apache : apache service activation] ***********************
ok: [inventaire-central]

TASK [apache-inventory : Apache configuration] *****************
changed: [inventaire-central]

RUNNING HANDLER [apache : apache restart] ***********************
changed: [inventaire-central]

PLAY RECAP *****************************************************
inventaire-central    : ok=6   changed=2   unreachable=0   failed=0
```

Ansible Galaxy

1. Présentation du site

Vous l'avez vu, les rôles sont un moyen de mettre en commun du code pour l'installation de programmes. Ansible Galaxy permet de répondre à une autre problématique : centraliser et trouver des rôles Ansible déjà écrits.

La première étape va consister à se rendre sur le site : https://galaxy.ansible.com/


Ansible Galaxy
galaxy.ansible.com
GALAXY
About
Help
Documentation
Login
Home
Search
Community
Home
Most Popular
System
Development
Networking
Cloud
Database
Monitoring
Packaging
Playbook Bundles
Security
Web
Download
Jump-start your automation project with great content from the Ansible community. Galaxy provides pre-packaged units of work known to
Share
Help other Ansible users by sharing the awesome roles and collections you create.
Featured
You're invited to AnsibleFest 2021!
Explore ways to automate, innovate, and accelerate at AnsibleFest 2021.
Join our free virtual event

Ce site met à disposition un moteur de recherche vous permettant de trouver les rôles mis à disposition par la communauté.

2. Recherche d'un rôle

Une autre solution pour rechercher un rôle est de passer par la commande ansible-galaxy en utilisant l'option search. Il est également possible de filtrer avec le nom d'un contributeur.

Pour cela, vous pouvez passer par l'option --author suivie du nom de l'auteur.

Vous rechercherez un rôle pour installer java publié par l'utilisateur viniciusfs.

Ci-dessous la commande correspondante :

```
$ ansible-galaxy search java --author viniciusfs
```

Ci-dessous le résultat de cette commande :

```
Found 1 roles matching your search:

Name              Description
----              -----------
viniciusfs.java Installs Java OpenJDK in CentOS/RHEL systems.
```

Avant l'installation d'un rôle, il est possible de récupérer quelques informations sur celui-ci avec l'option info suivie du nom du package à installer. Ci-dessous, le lancement pour récupérer des informations sur le rôle viniciusfs.java :

```
$ ansible-galaxy info viniciusfs.java
```

Et ci-dessous, un extrait des informations récupérées :

```
Role: viniciusfs.java
       description: Installs Java OpenJDK in CentOS/RHEL systems.
[...]

Role: viniciusfs.java
       description: Installs Java OpenJDK in CentOS/RHEL systems.
[...]
Installs Java OpenJDK in CentOS/RHEL systems.


## Role Variables

* `java_version`:
   - Description: OpenJDK version to install
   - Default: `1.7.0`


## Example Playbook

   - hosts: servers
     roles:
       - { role: viniciusfs.java }
[...]
```

3. Utilisation d'un rôle en provenance d'Ansible Galaxy

Vous allez tester un rôle permettant d'installer une version d'OpenJDK. Vous passerez pour cela par le rôle que vous venez de voir (viniciusfs.java). L'installation sera réalisée à l'aide de l'outil ansible-galaxy auquel vous passerez l'option install ainsi que le nom du rôle.

Sans autre option, ce rôle sera installé par défaut dans le répertoire /etc/ansible/roles.

Si vous n'avez pas les privilèges root, vous passerez l'option --roles-path suivie du chemin où vous installerez le rôle (vous utiliserez le répertoire roles). Ci-dessous la commande correspondante :

```
$ ansible-galaxy install viniciusfs.java -p roles
```

Ci-dessous, le résultat de cette commande :

```
- downloading role 'java', owned by viniciusfs
- downloading role from https://github.com/viniciusfs/
ansible-role-java/archive/master.tar.gz
- extracting viniciusfs.java to roles/viniciusfs.java
- viniciusfs.java (master) was installed successfully
```

Pour pouvoir utiliser ce fichier, vous devrez créer un playbook. Vous sauvegarderez ce playbook dans le fichier viniciusfs.java.yml et vous l'appliquerez au groupe de machine java.

Ci-dessous un exemple d'implémentation :

```
- name: "viniciusfs.java"
 hosts: java
 roles:
   - role: "viniciusfs.java"
```

4. Utilisation d'un fichier de prérequis

Vous êtes en mesure de récupérer des rôles en provenance d'Ansible Galaxy. Toutefois, l'installation de plusieurs rôles peut se révéler fastidieuse et vous n'avez pas forcément envie de tout stocker sur le site Ansible Galaxy.

C'est là où l'utilisation d'un fichier de prérequis au format YAML va vous permettre de répondre à ce type de questions.

Le fichier contient une liste des prérequis dans laquelle vous retrouverez les champs suivants :

- name : le nom du rôle à récupérer.

- src : la référence du rôle.

- version : la version à récupérer.

Pour le champ version, dans le cas d'une URL de type Git, il est possible de spécifier indifféremment une branche ou la clé de hachage d'un commit.

Par la suite, vous allez créer un fichier contenant deux références :

- une vers le rôle viniciusfs.java ;
- une autre vers le repository https://github.com/EditionsENI/ansible.

Ci-dessous le fichier requirements.yml correspondant à cette définition :

```
# Install a role from GitHub
- name: mediawiki
 src: https://github.com/Yannig/mediawiki-role

- src: viniciusfs.java
```

Il vous reste maintenant à fournir ce fichier à la commande ansible-galaxy avec les options install -r requirements.yml et l'emplacement où déposer les rôles (-p roles) :

```
$ ansible-galaxy install -r requirements.yml -p roles
```

SOLYTI

Ci-dessous le résultat de cette commande :

```
- extracting mediawiki to test/mediawiki
- mediawiki was installed successfully
- downloading role 'java', owned by viniciusfs
- downloading role from https://github.com/viniciusfs/
ansible-role-java/archive/master.tar.gz
- extracting viniciusfs.java to test/viniciusfs.java
- viniciusfs.java (master) was installed successfully
```

Dans le cas d'un projet complexe, il peut s'agir d'une excellente façon de gérer vos dépendances.

5. Comment lister les rôles d'Ansible Galaxy ?

Il est possible d'obtenir une liste des rôles installés sur la machine avec l'option list. Vous devez également ajouter l'option -p roles pour préciser l'emplacement où se trouvent les rôles en provenance de Galaxy.

La commande à lancer : `$ ansible-galaxy list -p roles`

Le résultat de cette commande : `- viniciusfs.java, master`

6. Suppression d'un rôle

Vous avez testé un rôle et ce dernier ne vous va pas ou tout simplement vous n'en avez plus l'utilité.

Vous pouvez à ce moment-là le supprimer avec le mot-clé remove suivi du nom du rôle à supprimer. Ci-dessous la commande pour supprimer le rôle viniciusfs.java :

```
$ ansible-galaxy remove viniciusfs.java -p roles
```

le résultat de cette commande :

```
- successfully removed viniciusfs.java
```

7. Collections Ansible

a. Origine du besoin

Ansible est un outil qui se veut universel. De fait, il embarque énormément de modules très divers :

- Red Hat maintient les modules génériques ou particulièrement importants (template, file, copy).
- Les sociétés fournissant de l'infrastructure dans les nuages (cloud) maintiennent des modules de gestion de leurs services (c'est notamment le cas pour Amazon/AWS ou Microsoft/Azure).

- Les sociétés proposant du matériel maintiennent les modules d'administration de composants physiques (baies de stockage, gestion réseaux/pare-feu/etc.).
- Enfin, la communauté peut s'occuper de certains modules tiers.

Cette organisation entraîne inévitablement des désagréments :

- Difficultés à tester les modules (notamment pour la gestion de matériels spécifiques).

- Nouveaux modules ayant besoin d'évoluer plus vite que le développement d'Ansible (nouveaux services cloud, modules expérimentaux).

- Modules spécifiques n'ayant pas forcément le niveau d'exigence nécessaire pour une inclusion dans les sources d'Ansible.

- Développement de composants propres à une entreprise.

- Utilisation de version de modules récents avec une plus ancienne version d'Ansible.

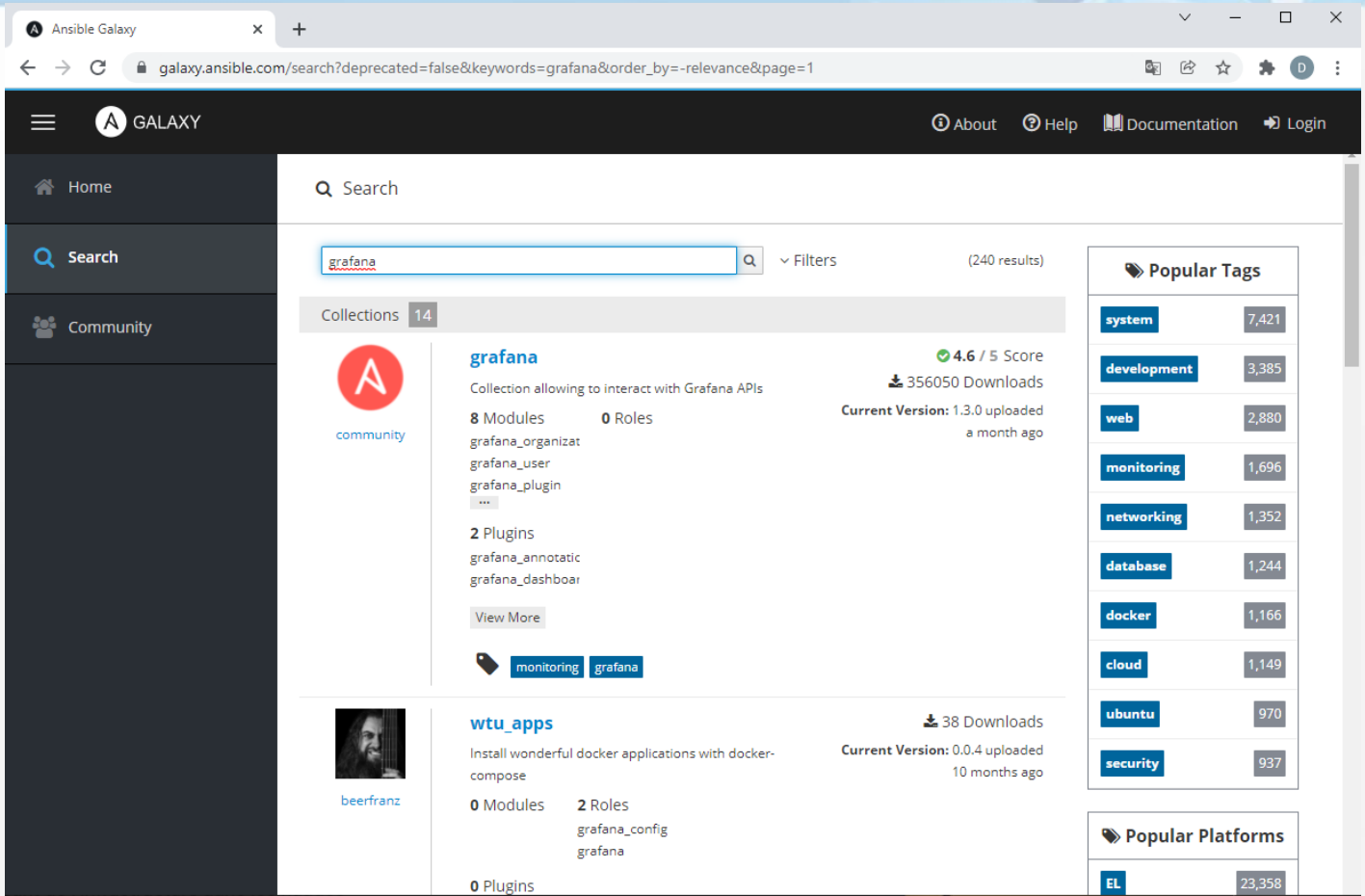
Afin de répondre à ces différents besoins, la notion de collection a été introduite en version 2.8 (tout d'abord sous forme d'expérimentation). Depuis la version 2.9, il s'agit d'une fonction stable.

b. Recherche de collections

Par la suite, nous nous intéresserons à l'installation d'une collection permettant l'administration de Grafana à l'aide d'Ansible.

La commande ansible-galaxy ne permet pas de faire de recherche. Pour cela, vous devez faire appel à l'adresse d'Ansible Galaxy : https://galaxy.ansible.com/

À la racine du site, sur la gauche de l'écran, cliquez sur le lien search. Dans le champ de recherche, saisissez la valeur « grafana ».


Ansible Galaxy
galaxy.ansible.com/search?deprecated=false&keywords=grafana&order_by=-relevance&page=1
GALAXY
About
Help
Documentation
Login
Home
Search
Community
Search
grafana
Filters
(240 results)
Collections 14
grafana
Collection allowing to interact with Grafana APIs
4.6 / 5 Score
356050 Downloads
Current Version: 1.3.0 uploaded a month ago
8 Modules
grafana_organizat
grafana_user
grafana_plugin
0 Roles
2 Plugins
grafana_annotatic
grafana_dashboar
View More
community
monitoring
grafana
wtu_apps
Install wonderful docker applications with docker-compose
38 Downloads
Current Version: 0.0.4 uploaded 10 months ago
0 Modules
2 Roles
grafana_config
grafana
0 Plugins
beerfranz
Popular Tags
system 7,421
development 3,385
web 2,880
monitoring 1,696
networking 1,352
database 1,244
docker 1,166
cloud 1,149
ubuntu 970
security 937
Popular Platforms
EL 23,358

Cliquez sur le premier lien de la recherche correspondant à la collection community.grafana afin d'en apprendre plus à son sujet.

c. Installation d'une collection

Une fois la collection déterminée, il devient possible de l'installer en faisant appel à la commande ansible-galaxy suivie des options suivantes :

- Le mot-clé collection.
- L'instruction install suivie de la collection à installer.

Dans le cas de l'installation de la collection community.grafana, la commande à lancer sera la suivante :

```
$ ansible-galaxy collection install community.grafana
```

Ci-dessous le résultat de cette commande :

```
Process install dependency map
Starting collection install process
Installing 'community.grafana:0.2.1' to '/home/yannig/.ansible/
collections/ansible_collections/community/grafana'
```

d. Consultation de la liste des collections

Afin de s'assurer que rien ne manque, il est possible de récupérer la liste des collections présentes. Pour cela, la commande accepte les instructions collection list.

Ci-dessous la commande complète :

```
$ ansible-galaxy collection list
```

Ci-dessous le résultat attendu lors de la présence des collections community.grafana et operator_sdk.util :

```
# /home/yannig/.ansible/collections/ansible_collections
Collection        Version
----------------- -------
community.grafana 0.2.1
operator_sdk.util 0.0.0
```

e. Utilisation d'une version spécifique

Lors de l'installation de la collection, l'utilisateur peut spécifier la version de la collection à installer en faisant suivre le nom de la collection par deux points (:) suivis de la version à installer. Ainsi, afin de réaliser l'installation de la version 0.2.1 de la collection community.grafana, vous devrez faire appel à la commande suivante :

```
$ ansible-galaxy collection install community.grafana:0.2.1
```

Il est également possible de spécifier des conditions sur ces versions à l'aide d'opérateurs. Ci-dessous la liste des opérateurs disponibles :

Opérateur	Exemple
`*` : n'importe quelle version	`my.collection:*`
`!=` : exclure une version	`my.collection:!=0.1.0`
`==` : sélection d'une version	`my.collection:==0.1.0`
`>=` : version supérieure ou égale	`my.collection:>=0.1.0`
`>` : version supérieure	`my.collection:>0.1.0`
`<=` : version inférieure ou égale	`my.collection:<=0.1.0`
`<` : version inférieure	`my.collection:<0.1.0`

Chaque condition peut être séparée par une virgule (,). Ci-dessous un exemple de sélection de la version supérieure à 0.1.0 et inférieure à 1.0.0 :

```
$ ansible-galaxy collection install "my.collection:>0.1.0,<1.0.0"
```

f. Utilisation d'un fichier de dépendances

La spécification des dépendances à télécharger peut également se faire à l'aide d'un fichier requirements.yml et du champ collections.

Ce champ se présente sous forme d'un tableau YAML contenant des enregistrements constitués des champs suivants :

- name : nom de la collection à télécharger.
- version : version à télécharger.

Ci-dessous un exemple de déclaration pour le téléchargement de la version 0.2.1 de la collection community.grafana :

```
collections:
 - name: community.grafana
   version: 0.2.1
```

SOLYTI

Sauvegardez cette déclaration dans le fichier requirements.yml. Testez ensuite l'installation à l'aide de la commande suivante :

```
$ ansible-galaxy collection install -r requirements.yml
```

g. Gestion des datasources Grafana

La collection est installée et il devient possible de s'en servir. Cette dernière embarque un module grafana_datasource. Ce module permet de définir une source de données pour Grafana à l'aide des champs suivants :

- name : nom de la source de données (exemple : test).
- grafana_url : adresse de l'interface Grafana à administrer (exemple : http://localhost:3000).

- grafana_user : nom de l'utilisateur Grafana (exemple : admin).
- grafana_password : mot de passe de l'utilisateur.
- ds_type : type de la source de données (exemple : prometheus).
- ds_url : adresse du moteur Prometheus (exemple : http://localhost:9090).

Au niveau de la définition du playbook, ce dernier respectera les éléments suivants :

- Application du playbook sur la machine localhost (hosts: localhost).
- Désactivation de la récupération des caractéristiques de la machine (gather_facts: no).
- Utilisation de la collection community.grafana (via le champ collections).

Ci-dessous un exemple de playbook reprenant toutes ces indications :

```
- name: "Define Prometheus datasource"
 hosts: localhost
 gather_facts: no
 collections:
   - community.grafana
 tasks:
   - name: "Define Prometheus datasource"
     grafana_datasource:
       name: "test"
       grafana_url: "http://localhost:3000"
       grafana_user: "admin"
       grafana_password: "admin"
       ds_type: "prometheus"
       ds_url: "http://localhost:9090"
```

Sauvegardez cette déclaration sous le nom de fichier define-datasource.yml.

Le lancement du playbook ne diffère pas de ce qui se fait d'habitude. Ci-dessous la commande à lancer :

```
$ ansible-playbook define-datasource.yml
```

Ci-dessous un extrait du lancement de ce playbook :

```
PLAY [Define Prometheus datasource] ***********************************

TASK [Define Prometheus datasource] ***********************************
changed: [localhost]
...
```

La source de données est maintenant prête.