COURS ANSIBLE 2

JOURS

Exercices

Version : 0.3
Date : 25/09/2024

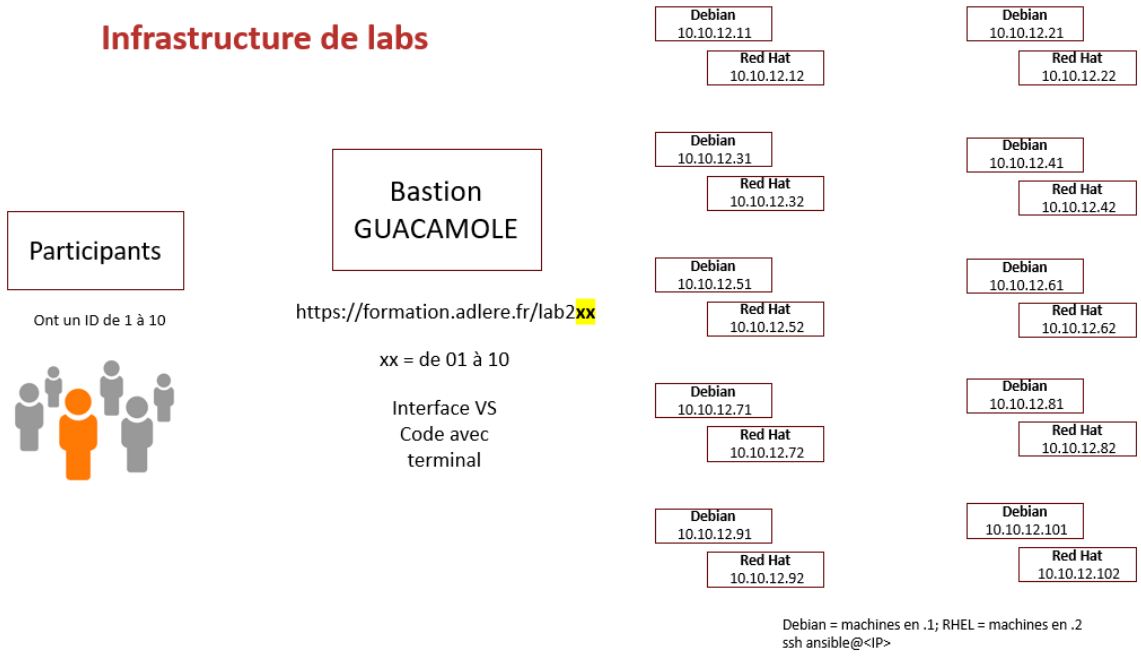
SOMMAIRE

1 INTRODUCTION

1.1 Historique des versions

Auteur	Version	Modification
Johan Klotz	1.11	Adaptation pour cours 3
Johan Klotz	1.1 02/04/2024	Adaptation pour cours 2
Johan Klotz	1.0 25/1/2024	Version initiale

1.2 Environnement d'exercice



2 PRESENTATION / PROCEDURE DE CONNEXION

Chaque élève travaille avec un compte non système sur une VM Linux bastion, avec un nom de type `lab2XX`, XX étant un numéro de 01 à 10.

Ex : lab209

Chaque élève dispose d'un accès à 2 VMs, sur lesquelles on peut se connecter avec un compte nommé ansible. La clef ssh de connexion est le fichier `~/.ssh/id_rsa`.

Chaque élève dispose d'une clef ssh qui est spécifique à ses deux VMs.

Les identifiants (noms, IP) des VMs de chaque élève sont donnés dans le précédent paragraphe, "1.2","Environnement d'exercice".

Action à faire :

➔ se connecter sur son environnement, on est alors dans son répertoire HOME.

Y créer un répertoire de travail, par exemple appelé '`playbooks`' :

```
$ mkdir -v playbooks
```

Se positionner dans ce répertoire :

```
$ cd playbooks
```

Le chemin d'accès de ce répertoire de travail est donc le suivant :

`/home/lab2xx/playbooks` (avec xx un numéro de 01 à 10).

Dans ce support, le répertoire HOME peut aussi être désigné par `$HOME`.

3 INSTALLATION ANSIBLE

On va procéder à différents façons d'installer Ansible.

Ansible est déjà installé sur la VM bastion, ces installations doivent se faire sur les 2 VMs des utilisateurs. Le code installé ne sera pas utilisé par le suite, il s'agit juste de démontrer comment le produit s'installe.

Les sorties d'écran affichées dans ce support ont été raccourcies (le bloc de caractères […] dénotant où les abréviations ont eu lieu).

3.1 sur la VM Red Hat

Connectez-vous sur la machine Red Hat (voir adresse IP en début de support), depuis le bastion, avec une commande ssh :

```
$ ssh ansible@IP
```

Passez root :

```
$ sudo su –
```

On va faire ici une installation d'Ansible au niveau OS.

Vérifier tout d'abord que la commande '`ansible`' n'est pas déjà installée :

```
# ansible
-bash: /usr/bin/ansible: Aucun fichier ou dossier de ce type
```

Afficher les dépôts disponibles :

```
$ dnf repolist
Utilisateur non « root », les référentiels de gestion des abonnements ne sont pas mis à jour
id du dépôt                                  nom du dépôt
rhel-8-for-x86_64-appstream-rpms             Red Hat Enterprise Linux 8 for x86_64 - AppStream (RPMs)
rhel-8-for-x86_64-baseos-rpms                Red Hat Enterprise Linux 8 for x86_64 - BaseOS (RPMs)
```

Premier essai, installation d'un package '`ansible`' :

```
$ sudo dnf install ansible
Mise à jour des référentiels de gestion des abonnements.
Dernière vérification de l'expiration des métadonnées effectuée il y a 3:29:25 le ven. 12 janv. 2024
18:47:02 CET.
Aucune correspondance pour le paramètre: ansible
Erreur : Impossible de trouver une correspondance: ansible
```

Cette erreur, qui est voulue, sert à montrer que sous RHEL 9 notamment, les outils Ansible sont dans un paquetage dénommé '`ansible-core`' et non '`ansible`'.

On installe ensuite ce paquetage '`ansible-core`' :

```
$ sudo dnf install ansible-core
```

```
Mise à jour des référentiels de gestion des abonnements.
Dernière vérification de l'expiration des métadonnées effectuée il y a 3:29:33 le ven. 12 janv. 2024
18:47:02 CET.
Dépendances résolues.
================================================================================================================
 Paquet                        Architecture  Version         Dépôt                              Taille
================================================================================================================
Installation:
 ansible-core                  x86_64   2.15.3-1.el8     rhel-8-for-x86_64-appstream-rpms   3.6 M
Installation des dépendances:
 git-core                      x86_64   2.39.3-1.el8_8   rhel-8-for-x86_64-appstream-rpms   11 M
 mpdecimal                     x86_64   2.5.1-3.el8      rhel-8-for-x86_64-appstream-rpms   93 k
 python3.11                    x86_64   3.11.5-1.el8_9   rhel-8-for-x86_64-appstream-rpms   30 k
 python3.11-cffi               x86_64   1.15.1-1.el8     rhel-8-for-x86_64-appstream-rpms   293 k
 python3.11-cryptography       x86_64   37.0.2-5.el8     rhel-8-for-x86_64-appstream-rpms   1.1 M
 python3.11-libs               x86_64   3.11.5-1.el8_9   rhel-8-for-x86_64-appstream-rpms   10 M
 python3.11-pip-wheel          noarch   22.3.1-4.el8     rhel-8-for-x86_64-appstream-rpms   1.4 M
 python3.11-ply                noarch   3.11-1.el8       rhel-8-for-x86_64-appstream-rpms   135 k
 python3.11-pycparser          noarch   2.20-1.el8       rhel-8-for-x86_64-appstream-rpms   147 k
 python3.11-pyyaml             x86_64   6.0-1.el8        rhel-8-for-x86_64-appstream-rpms   214 k
 python3.11-setuptools-wheel   noarch   65.5.1-2.el8   rhel-8-for-x86_64-appstream-rpms     720 k
 sshpass                       x86_64   1.09-4.el8   rhel-8-for-x86_64-appstream-rpms       30 k

Résumé de la transaction
================================================================================================================
================================================================================
Installer  13 Paquets

Taille totale des téléchargements : 29 M
Taille des paquets installés : 99 M
Voulez-vous continuer ? [o/N] : y
Téléchargement des paquets :
(1/13): sshpass-1.09-4.el8.x86_64.rpm
34 kB/s |  30 kB     00:00
(2/13): mpdecimal-2.5.1-3.el8.x86_64.rpm
89 kB/s |  93 kB     00:01
(3/13): python3.11-cffi-1.15.1-1.el8.x86_64.rpm
[…]
  Installation          : python3.11-ply-3.11-1.el8.noarch   7/13
  Installation          : python3.11-pycparser-2.20-1.el8.noarch   8/13
  Installation          : python3.11-cffi-1.15.1-1.el8.x86_64   9/13
  Installation          : python3.11-cryptography-37.0.2-5.el8.x86_64   10/13
  Installation          : python3.11-pyyaml-6.0-1.el8.x86_64   11/13
  Installation          : sshpass-1.09-4.el8.x86_64   12/13
  Installation          : ansible-core-2.15.3-1.el8.x86_64   13/13
  Exécution du scriptlet: ansible-core-2.15.3-1.el8.x86_64   13/13
  Vérification de       : sshpass-1.09-4.el8.x86_64   1/13
  Vérification de       : mpdecimal-2.5.1-3.el8.x86_64   2/13
  Vérification de       : python3.11-cffi-1.15.1-1.el8.x86_64   3/13
  Vérification de       : python3.11-ply-3.11-1.el8.noarch   4/13
  Vérification de       : python3.11-pycparser-2.20-1.el8.noarch   5/13
  Vérification de       : python3.11-pyyaml-6.0-1.el8.x86_64   6/13
  Vérification de       : python3.11-setuptools-wheel-65.5.1-2.el8.noarch   7/13
  Vérification de       : git-core-2.39.3-1.el8_8.x86_64   8/13
  Vérification de       : python3.11-cryptography-37.0.2-5.el8.x86_64   9/13
  Vérification de       : python3.11-pip-wheel-22.3.1-4.el8.noarch   10/13
  Vérification de       : ansible-core-2.15.3-1.el8.x86_64   11/13
  Vérification de       : python3.11-3.11.5-1.el8_9.x86_64   12/13
  Vérification de       : python3.11-libs-3.11.5-1.el8_9.x86_64   13/13
Produits installés mis à jour.

Installé:
  ansible-core-2.15.3-1.el8.x86_64         git-core-2.39.3-1.el8_8.x86_64                  mpdecimal-
2.5.1-3.el8.x86_64              python3.11-3.11.5-1.el8_9.x86_64
  python3.11-cffi-1.15.1-1.el8.x86_64      python3.11-cryptography-37.0.2-5.el8.x86_64     python3.11-
libs-3.11.5-1.el8_9.x86_64      python3.11-pip-wheel-22.3.1-4.el8.noarch
  python3.11-ply-3.11-1.el8.noarch         python3.11-pycparser-2.20-1.el8.noarch          python3.11-
pyyaml-6.0-1.el8.x86_64         python3.11-setuptools-wheel-65.5.1-2.el8.noarch
  sshpass-1.09-4.el8.x86_64

Terminé !
$
```

Vérifier en invoquant la commande '`ansible --version`' :

```
# ansible --version
ansible [core 2.14.9]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/ansible/.ansible/plugins/modules',
'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3.9/site-packages/ansible
  ansible collection location =
/home/ansible/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.9.18 (main, Jan  4 2024, 00:00:00) [GCC 11.4.1 20230605 (Red
Hat 11.4.1-2)] (/usr/bin/python3)
  jinja version = 3.1.2
  libyaml = True
```

L'installation est terminée sous Red Hat.

3.2 Debian 12

Pour Debian, on va effectuer une installation d'Ansible en tant que module Python dans un environnement virtual env. Cette procédure fonctionnerait aussi pour Red Hat.

Un virtual env est un environnement d'exécution et de développement de programmes Python isolé de l'environnement Python global : on peut y installer et désinstaller des composants dans une arborescence spécifique sans toucher aux modules Python présents au niveau OS.

Plus clairement, c'est une arborescence de répertoire où seront automatiquement installés et recherché des modules Python, car on fait pointer PATH dessus.

Tout d'abord, si ce n'est pas déjà fait, passer root, et mettre à jour la liste des packages disponibles dans les différents repo Debian.

```
$ sudo su -
# apt-get update
[…]
```

Installer ensuite le support des environnements virtuels (venv) pour Python :

```
# apt install python3.11-venv
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
Les paquets supplémentaires suivants seront installés :
  python3-pip-whl python3-setuptools-whl
Les NOUVEAUX paquets suivants seront installés :
  python3-pip-whl python3-setuptools-whl python3.11-venv
0 mis à jour, 3 nouvellement installés, 0 à enlever et 0 non mis à jour.
Il est nécessaire de prendre 2 835 ko dans les archives.
Après cette opération, 3 164 ko d'espace disque supplémentaires seront utilisés.
```

```
Souhaitez-vous continuer ? [O/n] o
```

Répondre O + ENTER (la sortie d'écran ci-dessous a été raccourcie).

```
Réception de :1 http://deb.debian.org/debian bookworm/main amd64 python3-pip-whl all 23.0.1+dfsg-1 [1 717 kB]
Réception de :2 http://deb.debian.org/debian bookworm/main amd64 python3-setuptools-whl all 66.1.1-1 [1 111 kB]
Réception de :3 http://deb.debian.org/debian bookworm/main amd64 python3.11-venv amd64 3.11.2-6 [5 892 B]
2 835 ko réceptionnés en 0s (42,9 Mo/s)
Sélection du paquet python3-pip-whl précédemment désélectionné.
(Lecture de la base de données... 46780 fichiers et répertoires déjà installés.)
[…]
Paramétrage de python3.11-venv (3.11.2-6) ...
#
```

On revient ensuite en tant qu'utilisateur ansible, et on crée un virtual env '`ansible-env`' :

```
# python3 -m venv ansible-env
```

Cela crée un sous-répertoire '`ansible-env`' dans le répertoire en cours, avec les liens et composants nécessaires pour l'exécution de programmes Python.

On va ensuite utiliser le module Pip pour installer Ansible.

Pip signifie "Pip installs package" ou "Preferred installation program". Il s'agit d'un gestionnaire de paquetages Python, dont le fonctionnement évolue à chaque version de Python.

Dans ce scénario, Pip est utilisé pour installer Ansible pour un utilisateur donné. En effet, c'est généralement une très mauvaise idée d'utiliser Pip en mode root et d'installer des paquetages Python au niveau du système, car c'est la quasi-assurance de casser un composant Python quelque part.

Noter que dans les versions récentes de Python, on ne peut rien installer en tant que root :

```
# pip3 install ansible
error: externally-managed-environment
× This environment is externally managed
╰─> To install Python packages system-wide, try apt install
    python3-xyz, where xyz is the package you are trying to
    install.
 […]
hint: See PEP 668 for the detailed specification.
```

Pour travailler ensuite dans le virtual env précédemment créé, taper la commande ci-dessous :

```
# . ./ansible-env/bin/activate
```

On installe ensuite Ansible dans ce virtual env (noter le préfixe '`(ansible-env)`' dans le prompt) :

```
(ansible-env) # pip3 install ansible
Collecting ansible
  Downloading ansible-9.4.0-py3-none-any.whl (46.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 46.4/46.4 MB 57.0 MB/s eta 0:00:00
Collecting ansible-core~=2.16.5
  Downloading ansible_core-2.16.5-py3-none-any.whl (2.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.3/2.3 MB 179.5 MB/s eta 0:00:00
Collecting jinja2>=3.0.0
  Downloading Jinja2-3.1.3-py3-none-any.whl (133 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 133.2/133.2 kB 110.0 MB/s eta 0:00:00
[…]
Installing collected packages: resolvelib, PyYAML, pycparser, packaging, MarkupSafe, jinja2, cffi,
cryptography, ansible-core, ansible
Successfully installed MarkupSafe-2.1.5 PyYAML-6.0.1 ansible-9.4.0 ansible-core-2.16.5 cffi-1.16.0
cryptography-42.0.5 jinja2-3.1.3 packaging-24.0 pycparser-2.22 resolvelib-1.0.1
```

On est toujours dans le virtual env, on contrôle en tapant la commande '`ansible --version`' :

```
# ansible --version
ansible [core 2.16.5]
  config file = None
  configured module search path = ['/home/ansible/.ansible/plugins/modules',
'/usr/share/ansible/plugins/modules']
  ansible python module location = /home/ansible/ansible/lib/python3.11/site-packages/ansible
  ansible collection location = /home/ansible/.ansible/collections:/usr/share/ansible/collections
  executable location = /home/ansible/ansible/bin/ansible
  python version = 3.11.2 (main, Mar 13 2023, 12:18:29) [GCC 12.2.0]
(/home/ansible/ansible/bin/python3)
  jinja version = 3.1.3
  libyaml = True
#
```

La version d'Ansible installée ainsi sera la plus récente possible pour la version de Python utilisée.

On utilise la commande '`deactivate`' pour sortir d'un virtual env :

```
(ansible-env) ansible@formateur-node1:[~]# deactivate
```

La commande '`ansible`' n'est alors plus accessible; il faut retourner dans le virtual env pour en disposer à nouveau :

```
ansible@formateur-node1:[~]# . ./ansible-env/bin/activate
```

Refaire à nouveau un '`deactivate`', puis se déconnecter du système.

Noter que les virtual env et Pip sont deux composants Python distincts et indépendants l'un de l'autre, simplement pour éviter de casser le fonctionnement de Python au niveau système, on préfèrera l'utiliser dans une virtual env.

Si on avait voulu, comme sur Red Hat, faire une installation d'Ansible au niveau système, avec des paquets préparés par l'équipe Debian, on aurait pu utiliser :

installation : '`apt install ansible`'

Désinstallation : '`apt autoremove`'

L'exercice est maintenant terminé.

4 CREATION D'UN INVENTAIRE

L'exercice va consister à créer un inventaire pour que chaque utilisateur puisse adresser ses VMs, et tester la connectivité.

Pour des raisons de démonstration, il y aura dans un premier temps des messages d'erreur de générés avant une correction finale qui réglera les problèmes.

4.1 Réalisation d'un inventaire

Rappel :

les opérations de ce chapitre prennent place dans une VM Linux dite 'bastion', dans un sous-répertoire '`playbooks`' du répertoire $HOME de l'utilisateur.

4.1.1 Fichier inventaire

➔ **adapter ci-dessous le X en fonction de son numéro d'utilisateur**

L'opérateur dispose de 2 systèmes qu'il peut utiliser comme cible de ses actions :

```
lab2XX-node1
lab2XX-node2
```

XX correspond au nombre associé à l'identifiant d'utilisateur.

Les noms ci-dessus ne sont pas des noms réseau (ie défini avec une commande hostname ou équivalente), ce sont juste des identifiants que l'on va utiliser dans un inventaire Ansible.

Par exemple, si on est l'utilisateur lab7, les VMs dont on dispose sont nommées :

```
lab207-node1
lab207-node2
```

Dans le répertoire de travail, créer un fichier de nom `inventaire`, avec pour contenu :

```
[deb]
lab2XX-node1

[rhel]
```

```
lab2XX-node2

[infra:children]
deb
rhel
```

Tester la connexion aux systèmes de l'inventaire avec la commande ad-hoc suivante :

```
$ ansible -i inventaire -m ping all
```

➔ La commande va échouer avec un message similaire à celui ci-dessous, pouvez-vous déterminer pourquoi ?

```
lab2XX-node1 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: ssh: Could not resolve hostname
lab2XX-node1: Name or service not known",
    "unreachable": true
}
lab2XX-node2 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: ssh: Could not resolve hostname
lab10-node2: Name or service not known",
    "unreachable": true
}
```

Ne pas chercher à corriger par soi-même, continuer l'exercice.

Éditer à nouveau le fichier '`inventaire`', et le compléter en rajoutant des informations IP.

Reprendre l'exemple ci-dessous mais adapter avec les informations IP appropriées.

```
[deb]
lab2XX-node1       ansible_host=10.10.12.X1

[rhel]
lab2XX-node2       ansible_host=10.10.12.X2

[infra:children]
deb
rhel
```

Tester à nouveau la connexion aux systèmes de l'inventaire avec la commande ad-hoc suivante :

```
$ ansible -i inventaire -m ping all
```

➔ La commande va échouer avec un message similaire à celui ci-dessous, pouvez-vous déterminer pourquoi ? comment a évolué la situation avec le précédent essai ?

```
lab2XX-node1 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: formation@10.10.10.11:
Permission denied (publickey,gssapi-keyex,gssapi-with-mic,password).",
```

```
    "unreachable": true
}
lab2XX-node2 | UNREACHABLE! => {
    "changed": false,
    "msg": "Failed to connect to the host via ssh: formation@10.10.10.12:
Permission denied (publickey,gssapi-keyex,gssapi-with-mic,password).",
    "unreachable": true
}
```

On va voir au prochain paragraphe comment régler ce problème, en développant une configuration de connexion appropriée dans le fichier ansible.cfg.

4.1.2 Fichier ansible.cfg

Le fichier de configuration '**`ansible.cfg`**' va permettre notamment de paramétrer la connexion avec les systèmes gérés par Ansible.

On peut aussi l'utiliser pour définir un nom d'inventaire par défaut, pour ne pas avoir à désigner celui-ci systématiquement sur la ligne de commande, avec l'option '`-i inventaire`', comme dans '`ansible-playbook -i inventaire […]`'.

Dans le répertoire de travail, on crée donc '`ansible.cfg`,' avec le contenu suivant :

```
[defaults]
inventory = /home/USER/playbooks/inventaire
remote_user = ansible
host_key_checking = false
private_key_file = /home/USER/.ssh/id_rsa
```

➔ adapter USER en fonction de son identifiant (lab2XX); modifier au besoin le nom du répertoire '`playbooks`' si on a choisi un nom différent comme répertoire de travail.

Dans ce fichier, on a spécifié le nom du compte de connexion et sa clef ssh, pour résoudre les problèmes rencontrés au point précédent.

Relancer la commande '`ansible -m ping`', mais sans spécifier d'inventaire cette fois :

```
$ ansible -m ping all
```

La commande doit maintenant réussir :

```
lab2XX-node2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
lab2XX-node1 | SUCCESS => {
    "ansible_facts": {
```

```
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
```

Si ce n'est pas le cas, analyser le message d'erreur pour comprendre ce qui est en cause.

IMPORTANT : Cette commande '`ansible -m ping`' n'effectue pas un ping au sens traditionnel du terme. Elle effectue une connexion sur les systèmes cibles, et affiche quel interpréteur Python y a été identifié. Elle permet donc de valider en une seule fois :

- que les cibles sont bien joignables sur le réseau
- que les identifiants de connexion sont fonctionnels
- que la cible dispose d'un interpréteur Python

À ce stade, on a donc défini un inventaire et une méthode de connexion fonctionnels.

Le compte de connexion a été spécifié dans le fichier '`ansible.cfg`'; il aurait pu l'être dans l'inventaire, par exemple avec le bloc ci-dessous :

```
[all:vars]
ansible_user=ansible
```

Le support de cours présente d'autres paramètrages que l'on peut positionner dans le fichier de configuration ou le fichier d'inventaire.

4.2 « Host patterns »

Le terme de 'host pattern' désigne l'expression utilisée sur la ligne de commande ou dans un playbook, pour désigner les systèmes gérés qui vont être la cible d'une automatisation.

Ces 'host patterns' sont construits en fonction des systèmes et groupes définis dans le fichier d'inventaire.

Ci-dessous, on va créer un inventaire fictif; les machines listées n'existent pas dans l'environnement d'exercice, ce n'est pas nécessaire pour la démonstration, il n'y aura aucune réelle tentative de connexion dessus.

1. Créer dans le répertoire de travail un fichier `inventaire_demo1` tel que ci-dessous :

```
srv1.example.com
srv2.example.com

[web]
nginx01.example.com
nginx02.example.com
```

```
[db]
db1.example.com
db2.example.com
db3.example.com

[idf]
db1.example.com
nginx01.example.com

[paca]
db2.example.com
db3.example.com
nginx02.example.com

[dev]
nginx01.example.com
db3.example.com

[prod]
nginx01.example.com
db1.example.com
db2.example.com

[function:children]
web
db

[region:children]
idf
paca

[environments:children]
dev
prod

[ip]
192.168.1.10
192.168.1.20
```

Créer un playbook `pattern.yml`, tel que ci-dessous. Il utilise le module `ansible.builtin.debug` pour afficher le nom de chaque cible sur laquelle il est invoqué :

```
---
- name: Affiche le nom de la cible du playbook
  hosts: all
  gather_facts: no

  tasks:
  - name: Affichage du nom
    ansible.builtin.debug:
      msg: "{{ inventory_hostname }}"
```

Ensuite, pour répondre à différents challenges, on va utiliser et faire varier la commande ci-dessous :

```
$ ansible-playbook -i inventaire_demo1 pattern.yml -l ip
```

Ici, l'option clef qui permet de faire varier la cible est l'option `-l` qui est ensuite suivie d'un argument permettant de spécifier le motif définissant le groupe ou les systèmes qui doivent faire l'objet de l'application du playbook (ci-dessus, le groupe 'ip' est désigné).

En s'inspirant de la ligne de commande précédemment donnée en exemple, relancer l'exécution du playbook `pattern.yml` de façon à cibler :

- les machines du groupe prod
- les machines des groupes paca et idf
- la liste des machines dont l'adresse ip commence par 192
- la liste des machines dont le nom ne contient pas un 2
- les machines dont le nom contient un a, mais pas le chiffre 1
- les machines à la fois dans le groupe dev et idf

On peut s'aider de la documentation Ansible sur le sujet :

https://docs.ansible.com/ansible/latest/inventory_guide/intro_patterns.html

Solutions

```
ansible-playbook -i inventaire_demo1 pattern.yml -l prod
```

```
ansible-playbook -i inventaire_demo1 pattern.yml -l paca,idf
```

```
ansible-playbook -i inventaire_demo1 pattern.yml -l 192*
```

À partir d'un certain degré de sophistication, il vaut mieux utiliser des guillemets simple pour ne pas interférer avec le shell :

```
ansible-playbook -i inventaire_demo1 pattern.yml -l '!*2*'
```

```
ansible-playbook -i inventaire_demo1 pattern.yml -l '*a*,!*1*'
```

```
ansible-playbook -i inventaire_demo1 pattern.yml -l 'dev:&idf'
```

L’exercice est maintenant terminé.

5 COMMANDES AD-HOC

NOTE IMPORTANTE :

Pour un meilleur apprentissage, il est recommandé de taper complètement les commandes et de ne pas procéder par copier/coller.

A ce stade, on a donc un inventaire fonctionnel, et un fichier ansible.cfg fonctionnel, de tel façon que si on tape :

$ ansible -m ping all

… les 2 vms répondent correctement.

On va donc ici utiliser un certain nombre de commandes ad-hoc pour agir sur les systèmes distants :

Afficher l'identifiant utilisé par Ansible :

```
$ ansible -m command all -a "whoami"
```

Ou encore le nom de la machine :

```
$ ansible -m command all -a "hostname"
```

Noter que le module `command` est implicite si aucun module n'est explicitement désigné avec `-m` :

```
$ ansible all -a "whoami"
```

Vérifier le compte utilisé pour l'escalade de privilèges :

```
$ ansible all -a "whoami" -b
```

Vérifier l'état des systèmes de fichiers :

```
$ ansible all -a "df -Th"
```

Tester le '|' dans le module command, constater qu'il ne marche pas :

```
$ ansible all -a "df -Th | grep /dev/sda1"
```

Avec le module shell, ça passe :

```
$ ansible all -m shell -a "df -Th | grep /dev/sda1"
```

Afficher tous les facts et parcourir le contenu disponible :

```
$ ansible all -m setup | less
```

On va maintenant travailler avec les 'Ansible Facts'

Afficher les informations sur la mémoire totale :

```
$ ansible all -m ansible.builtin.setup -a 'filter=ansible_memtotal_mb'
```

La même commande mais avec l'option -o, pour améliorer un peu l'affichage :

```
$ ansible all -m ansible.builtin.setup -a 'filter=ansible_memtotal_mb' -o
```

Afficher la mémoire libre:

```
$ ansible all -m ansible.builtin.setup -a 'filter=ansible_memfree_mb'
```

Afficher toutes les informations sur la mémoire :

```
$ ansible all -m ansible.builtin.setup -a 'filter=ansible_*_mb'
```

Comparer avec la sortie de la commande 'free -m' :

```
$ ansible all -a "free -m"
```

Récupérer l'ID de machine :

```
$ ansible all -m ansible.builtin.setup -a 'filter=ansible_machine_id'
```

Afficher la distribution (Debian, RedHat, ...) :

```
$ ansible all -m ansible.builtin.setup -a 'filter=distribution'
```

Afficher l'identifiant de la version :

```
$ ansible all -m ansible.builtin.setup -a 'filter=distribution_version'
```

Afficher l'identifiant majeur de la version :

```
$ ansible all -m ansible.builtin.setup -a 'filter=distribution_major_version'
```

Afficher les informations IP relatives à l'interface réseau par défaut :

```
$ ansible all -m ansible.builtin.setup -a 'filter=default_ipv4'
```

On va maintenant utiliser quelques modules Ansible

Recopier sur la machine distante le fichier /etc/hosts de votre système Ansible :

```
$ ansible -m copy all -a "src=/etc/hosts dest=/tmp/myhosts mode='0400'"
```

Contrôler :

```
$ ansible all -a "ls -al /tmp/myhosts"
```

Installer chrony :

```
$ ansible -m apt all -a"name=chrony state=present"
```

Cela ne va pas marcher, ou alors partiellement, et ce pour deux raisons :

- il faut être root
- les machines ont des OS différents, Red Hat et Debian; il faut donc utiliser un module plus générique, en l'occurrence package.

La bonne commande est donc :

```
$ ansible -m package all -a "name=chrony state=present" -b
```

Procéder à l'installation de git :

```
$ ansible all -m package -a "name=git state=present" -b
```

Comparer les versions de git :

```
$ ansible all -a "git --version"
```

Procéder au clone d'un repository git :

```
$ ansible all -m git -a "repo=https://github.com/klotzjkl/docs dest=/tmp/docs
update=yes version=main"
```

Afficher le fichier précédemment cloné :

```
$ ansible all -m command -a "cat /tmp/docs/lorem.md"
```

La dernière ligne de ce fichier commence par "Ut a diam …", nous allons la remplacer par une ligne de copyright :

```
$ ansible all -m lineinfile -a 'state=present regex="^Ut" line="(c) Lipsum.com"
path=/tmp/docs/lorem.md'
```

Ré-afficher le fichier précédemment cloné pour vérifier le changement :

```
$ ansible all -m command -a "cat /tmp/docs/lorem.md"
```

En fin de ce fichier, on va insérer le contenu de la variable "ansible_host" (c'est une variable dite magique, qui existe toujours) :

```
$ ansible all -m lineinfile -a 'state=present line="\n\nSYSTEM {{
inventory_hostname }} ({{ansible_host}})" path=/tmp/docs/lorem.md insertafter=EOF'
```

➔ Noter l'utilisation des guillemets simples pour encadrer l'argument de l'option -a, et des guillemets doubles à l'intérieur pour encadrer des chaînes de caractères.

Taper l'exemple ci-dessous, où on passe une variable à la volée :

```
$ ansible all -m lineinfile -a 'state=present line="Fichier modifié par {{ auteur
}}\n**********\n\n" path=/tmp/docs/lorem.md insertbefore=BOF' -e "auteur='John
DOE'"
```

➔ Ici on a utilisé des guillemets simples à l'intérieur de guillemets doubles, eux-mêmes contenus dans une expression encadrée par des guillemets simples;

➔ le `-e` sert à passer des variables (appelées extra-vars) au module appelé

La précédente modification ayant inséré du texte en début du fichier, utiliser `'head'` pour afficher les premières lignes :

```
$ ansible all -m shell -a "cat /tmp/docs/lorem.md | head -2"
```

On récupère en local le fichier :

```
$ ansible all -m fetch -a "src=/tmp/docs/lorem.md dest=lorem_files flat=no"
```

➔ Noter que les fichiers sont stockés dans le sous-répertoire lorem_files, avec une arborescence basée sur le nom de chaque machine.

On finit l'exercice en effaçant le répertoire précédemment créé lors du clone :

```
$ ansible all -m file -a "path=/tmp/docs state=absent"
```

Quelles erreurs peut-on faire avec cette commande `ansible` ?

- utilisation de mauvais identifiants ou oubli de l'escalade de privilèges
- contacter les mauvaises machines
- utilisation d'un module (-m XXX) qui n'existe pas (il faut donc valider qu'il existe et se le procurer)
- l'oubli de `-a` devant les arguments

L'exercice est maintenant terminé.

6 VARIABLES

Créer un playbook affichant simplement un message, que l'on utilisera comme base pour travailler en local avec les variables.

➔ **Ne pas faire de copier-coller mais le taper**.

```
---
- name: Mon playbook
  hosts: localhost
  connection: local
  gather_facts: no

  tasks:
  - name: Affiche un message
    ansible.builtin.debug:
      msg: "Message"
```

En s'inspirant du support de cours (slide 7 du chapitre sur les variables), modifier le playbook pour définir en en-tête les variables suivantes :

jour: jeudi

mois: avril

➔ Ajouter une tâche `debug` pour afficher la variable jour, et une tâche `debug` pour afficher la variable mois, soit 2 tâches `debug` à écrire.

➔ Invoquer à nouveau le playbook, mais cette-fois en indiquant en ligne de commande (i.e extra-vars) une nouvelle valeur pour les variables jour et mois.

➔ Créer un sous-répertoire `files`, y placer un fichier '`variables.yml`', dans lequel les variables suivantes sont définies:

semaine: 'Du lundi au vendredi'

weekend: 'samedi et dimanche'

➔ Créer un playbook qui prend en entrée (i.e extra-vars) ce fichier `files/variables.yml`, et comporte une seule tâche affichant ces deux variables.

➔ Modifier le playbook, cette-fois-ci charger le fichier de variables `files/variables.yml` directement depuis le playbook, sans extra-vars.

➔ Modifier le playbook, mettre '`gather_facts: yes`'

Modifier le playbook pour qu'il n'y ait plus qu'une seule tâche, qui affiche :

- la version de python sur la machine locale

- le nombre de processeurs

- la quantité de mémoire totale

Astuce: pour identifier la variable sur la version de Python, explorez la sortie de la commande '`ansible -m setup localhost,`'. Les deux autres sont dans le support de cours.

➔ Challenge : modifier le playbook pour qu'il affiche les ansible_facts des machines de l'inventaire, et non de localhost comme dans l'exemple.

Exemple de solution

```
---
- name: Affiche des ansible_facts spécifiques
  hosts: localhost
  connection: local
  gather_facts: yes
  tasks:
    - name: Affichage de facts
      ansible.builtin.debug:
        msg: "Host {{ ansible_facts['fqdn'] }}, Python {{
ansible_facts['python_version'] }}, {{ ansible_facts['processor_count'] }}
processeurs et {{ ansible_facts['memtotal_mb'] }} MiB de mémoire totale."
```

L’exercice est maintenant terminé.

7 CREATION DE PLAYBOOKS

Plusieurs exercices sont proposés. le premier consiste à créer des playbooks pour installer / désinstaller un service Apache pour environnement Red Hat, le second pour faire de même environnement mixte Red Hat / Debian, le troisième pour créer un compte d'administration.

7.1 Apache pour RHEL

NOTE IMPORTANTE : ce playbook devra uniquement être exécuté sur la machine Red Hat (utiliser l'option '`-l <MACHINE RHEL>`' avec la commande '`ansible-playbook`' qui va l'exécuter).

7.1.1 Directives

Le premier playbook à écrire va donc permettre de procéder à l'installation et configuration basique d'un service Apache, de la façon suivante :

- installation du paquetage de firewalld (firewalld)
- installation du paquetage Apache (httpd)
- création du répertoire qui va accueillir des pages web
- démarrage du service Apache
- démarrage du service firewalld
- Création d'une page index.html avec un message personnalisé
- Paramétrage du firewall pour qu'il autorise les connexion sur le port 80

Il faut bien écrire 7 tâches dans le playbook (on pourrait l'écrire différemment et optimiser, mais pour l'instant il est demandé de coller à cette structure).

Ce playbook s'appellera '`apache-install-rhel.yml`'.

Conditions de succès :

- depuis la machine bastion où le playbook est développé et exécuté, la commande `curl` doit permettre d'afficher le contenu du fichier index.html, par exemple :

  ```
  $ curl http://ADRESSE_IP
  ```

  doit renvoyer :

  ```
  Hello World !
  ```

(si le contenu de `index.html` a été positionné sur 'Hello World !').

On va ensuite écrire un deuxième playbook, complémentaire du premier, qui doit procéder à la désinstallation de tous les composants installés et configurés par le premier playbook :

- désinstallation des paquetages httpd et firewalld
- effacement du répertoire /var/www et de son contenu

Ce playbook s'appellera '`apache-uninstall-rhel.yml`'.

Conditions de succès :

- le service ne répond plus à la commande curl

Optimisation du playbook

Reprendre le playbook d'installation d'Apache, en l'optimisant et en utilisant des variables, de telle façon que :

- un seule tâche suffise pour installer les deux paquetages httpd et firewalld; les noms des paquetages font l'objet de variables
- une seule tâche suffise pour démarrer et activer à chaque démarrage système les services httpd et firewalld
- mettre en variable le chemin du fichier html contenant le message, ainsi que le nom de ce fichier (donc deux variables à définir et utiliser)

7.1.2 Exemple de solution

Playbook d'installation Apache

```
---
- hosts: all
  tasks:

  - name: Installation du firewall
    ansible.builtin.dnf:
      name: firewalld
      state: present
    become: true

  - name: Installation Apache
    ansible.builtin.dnf:
      name:
       - httpd
       - tree
      state: present
    become: true

  - name: Création du répertoire de pages web
    ansible.builtin.file:
      path: /var/www/html
      state: directory
      mode: '0644'
      owner: apache
      group: root
    become: true

  - name: Démarrage et activation  Apache
    ansible.builtin.service:
      name: httpd
      state: started
      enabled: true
    become: true

  - name: Démarrage et activation Firewalld
    ansible.builtin.service:
      name: firewalld
      state: started
      enabled: true
    become: true

  - name: Création page d'index
    ansible.builtin.copy:
      dest: /var/www/html/index.html
      content: "Hello World !\n"
      owner: apache
      group: root
      mode: '0644'
    become: true

  - name: Paramétrage firewalld
```

```
    ansible.posix.firewalld:
      service: http
      permanent: true
      state: enabled
      immediate: true
    become: true
```

Playbook de désinstallation

Donner à ce playbook le nom de '`apache-uninstall-rhel.yml`'.

```
---
- hosts: all
  tasks:

  - name: Désinstallation du firewall
    ansible.builtin.dnf:
      name: firewalld
      state: absent
    become: true

  - name: Désinstallation Apache
    ansible.builtin.dnf:
      name: httpd
      state: absent
    become: true

  - name: Purge répertoire /var/www
    ansible.builtin.file:
      path: /var/www
      state: absent
    become: true
```

Exemple de playbook utilisant des variables :

```
---
- hosts: all
  gather_facts: no
  vars:
    http_pkg: httpd
    firewall_pkg: firewalld
    chemin: '/var/www/html'
    fichier: 'index.html'

  tasks:

  - name: Installation des services
    ansible.builtin.dnf:
      name:
        - "{{ http_pkg }}"
        - "{{ firewall_pkg }}"
      state: present
    become: true

  - name: Création répertoire  {{ chemin }}
```

```
    ansible.builtin.file:
      path: "{{ chemin }}"
      state: directory

  - name: Création page d'index
    ansible.builtin.copy:
      dest: "{{ chemin }}/{{ fichier }}"
      content: "Hello World !\n"
      owner: apache
      group: root
      mode: '0644'
    become: true

  - name: Démarrage et activation des services
    ansible.builtin.service:
      name: "{{ item }}"
      state: started
      enabled: true
    loop:
      - "{{ http_pkg }}"
      - "{{ firewall_pkg }}"
    become: true

  - name: Paramétrage firewalld
    ansible.posix.firewalld:
      service: http
      permanent: true
      state: enabled
      immediate: true
    become: true
```

7.2 Apache pour environnements mixtes

On va ici modifier le même playbook, pour qu'il prenne en charge aussi la machine Debian en utilisant des conditions avec le mot-clef '`when:`', et utiliser des variables pour personnaliser la page '`index.html`' générée.

7.2.1 Directives

➔ Tout d'abord, recopier le playbook ' `apache-install-rhel.yml` ' sous le nom ' `apache-install-mixte.yml` '.

Il va falloir modifier le playbook, pour qu'il effectue les mêmes opérations sur les machines, mais en gardant à l'esprit que Debian diffère de Red Hat dans notre cas, sur les points suivants :

- le paquetage à installer s'appelle apache2 sur Debian

- l'utilisateur sous lequel tourne Apache s'appelle www-data sur Debian
- le service à démarrer s'appelle apache2 sous Debian

Le fichier index.html généré sur les cibles doit renvoyer un bloc d'informations systèmes comme ci-dessous :

```
# curl http://10.10.10.252
Hello ici système formateur-node2
Nom long: formateur-node2.adlere.local
Adresse IP : 10.10.10.252
Distribution : RedHat, 9.3
```

NOTES IMPORTANTES :

1. Il ne faudra pas procéder à l'installation ni configuration de firewall sur aucun des deux systèmes.
2. On continue à utiliser le module **`ansible.builtin.copy`** pour générer le fichier cible.

NOTES UTILES :

1. On peut utiliser la séquence \n dans une chaîne de texte pour générer un retour à la ligne.
2. Les variables à utiliser sont dans les slides sur les variables (Ansible Facts, chapitre 4)

Conditions de succès :

Lorsque le service Apache est interrogé par '`curl`', (`curl http://labX-node1`), il doit répondre par une page d'information renvoyant :

- le nom court de la machine
- son nom long
- son adresse IP par défaut
- sa distribution (Red Hat ou Debian)
- ses informations de version (9.3 par exemple)

Challenges optionnels :

Exercice déjà fini ? bravo, maintenant

- modifier le playbook de façon à utiliser le module **`ansible.builtin.template`** pour générer le fichier index.html au lieu de **`ansible.builtin.copy`**
- écrire un playbook avec le module **`ansible.builtin.uri`** pour interroger les 2 machines, au lieu de taper manuellement une commande curl

7.2.2 Exemple de solution

La solution proposée est volontairement laide, elle permet d'illustrer simplement 2 choses :

- d'un OS à l'autre, il peut y avoir des différences difficilement réconciliables
- la possibilité d'utiliser le mécanisme de condition avec 'when' pour diriger quand une tâche est exécutée ou non

```
---
- hosts: all
  gather_facts: true

  tasks:

  - name: Installation Apache si REDHAT
    ansible.builtin.dnf:
      name:
       - httpd
       - tree
      state: present
    become: true
    when: ansible_facts['distribution'] == "RedHat"

  - name: Installation Apache si Debian
    ansible.builtin.apt:
      name:
       - apache2
       - tree
      state: present
    become: true
    when: ansible_facts['distribution'] == "Debian"

  - name: Création du répertoire de pages web si REDHAT
    ansible.builtin.file:
      path: /var/www/html
      state: directory
      mode: '0755'
      owner: apache
      group: root
    become: true
    when: ansible_facts['distribution'] == "RedHat"

  - name: Création du répertoire de pages web si DEBIAN
    ansible.builtin.file:
      path: /var/www/html
      state: directory
      mode: '0755'
      owner: www-data
      group: root
    become: true
    when: ansible_facts['distribution'] == "Debian"



  - name: Création page d'index si REDHAT
    ansible.builtin.copy:
      dest: /var/www/html/index.html
      content: "Hello ici système {{ ansible_facts['hostname'] }}\nNom long: {{
ansible_facts['fqdn'] }}\nAdresse IP : {{ ansible_facts['default_ipv4']['address']
}}\nDistribution      :      {{      ansible_facts['distribution']      }},      {{
ansible_facts['distribution_version'] }}\n\n"
      owner: apache
      group: root
```

```
      mode: '0644'
    become: true
    when: ansible_facts['distribution'] == "RedHat"

  - name: Création page d'index si DEBIAN
    ansible.builtin.copy:
      dest: /var/www/html/index.html
      content: "Hello ici système {{ ansible_facts['hostname'] }}\nNom long: {{
ansible_facts['fqdn'] }}\nAdresse IP : {{ ansible_facts['default_ipv4']['address']
}}\nDistribution      :      {{      ansible_facts['distribution']      }},      {{
ansible_facts['distribution_version'] }}\n\n"
      owner: www-data
      group: root
      mode: '0644'
    become: true
    when: ansible_facts['distribution'] == "Debian"

  - name: Démarrage et activation Apache si RedHat
    ansible.builtin.service:
      name: httpd
      state: started
      enabled: true
    become: true
    when: ansible_facts['distribution'] == "RedHat"

  - name: Démarrage et activation Apache si DEBIAN
    ansible.builtin.service:
      name: apache2
      state: started
      enabled: true
    become: true
    when: ansible_facts['distribution'] == "Debian"
```

Playbook de désinstallation

```
---
- hosts: all
  gather_facts: true

  tasks:

  - name: Désinstallation Apache si REDHAT
    ansible.builtin.dnf:
      name: httpd
      state: absent
    become: true
    when: ansible_facts['distribution'] == "RedHat"

  - name: Désinstallation Apache si Debian
    ansible.builtin.apt:
      name: apache2
      state: absent
    become: true
    when: ansible_facts['distribution'] == "Debian"

  - name: Purge répertoire /var/www
    ansible.builtin.file:
      path: /var/www
      state: absent
```

```
    become: true
```

Interroger les serveurs webs avec un playbook plutôt que la commande curl :

```
---
- hosts: all
  gather_facts: false

  tasks:
    - name: Requête sur les serveurs web …
      ansible.builtin.uri:
        url: "http://{{ ansible_host }}/index.html"
        return_content: yes
      register: this

    - name: Affiche le résultat de la requête
      ansible.builtin.debug:
        var: this.content
```

8 CREATION D'UN COMPTE D'ADMINISTRATION SUR RHEL

8.1 Directives

L'objet est de créer un compte de connexion doté d'une clef ssh spécifique.

Ce playbook est uniquement à destination d'une machine de type Red Hat.

Tout d'abord, en local et en ligne de commande, créer une clef de type ed25519 dans un fichier `sysadm` (utiliser la commande `ssh-keygen`).

```
$ ssh-keygen -t ed25519 -f sysadm
```

Écrire un playbook qui effectue les tâches ci-dessous :

- passe le SELinux en mode permissif
- création d'un un groupe sysadm, gid 1024
- création d'un compte sysadm, uid 1024, home /sysadm
- mettre PermitRootLogin à no dans /sshd_config
- injecter la clef ssh de sysadm
- attribuer des droits root dans le fichier sudoers
- redémarrer le service sshd

Utiliser un handler pour redémarrer le service sshd.

Vérification :

- le playbook s'exécute sans erreur
- une connexion sysadm avec la clef ssh fonctionne
- une fois connecté, la commande id permet d'identifier les paramètres du compte, et `echo $HOME` renvoie le répertoire attendu

Par exemple :

```
$ id
uid=1024(sysadm) gid=1024(sysadm) groupes=1024(sysadm),10(wheel)
contexte=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
$ echo $HOME
/sysadm
```

8.2 Exemple de solution

Tout d'abord, en local, utiliser la commande ci-dessous pour générer une paire de clefs ssh :

```
$ ssh-keygen -t ed25519 -f sysadm
```

Une clef privée (`sysadm`) ainsi qu'une clef publique (`sysadm.pub`) sont générées.

```
---
- name: Injection du compte sysadm
  hosts: all
  gather_facts: no
  become: yes

  tasks:
  - name: SELinux en mode permissif
    ansible.posix.selinux:
      policy: targeted
      state: permissive

  - name: Création du groupe
    ansible.builtin.group:
      name: sysadm
      state: present
      gid: 1024

  - name: Création de l'utilisateur sysadm
    ansible.builtin.user:
      name: sysadm
      uid: 1024
      home: /sysadm
      group: sysadm
      groups: wheel
      append: yes

  - name: Désactivation connexion root ssh
    ansible.builtin.lineinfile:
      path: /etc/ssh/sshd_config
      line: "PermitRootLogin no"
      backup: yes
      regexp: ^PermitRootLogin
    notify: Restart ssh

  - name: Clef ssh
    ansible.posix.authorized_key:
      user: sysadm
      state: present
      key: "{{ lookup('file','./sysadm.pub') }}"

  handlers:
    - name: Restart ssh
      ansible.builtin.service:
        name: sshd
        state: restarted
```

L’exercice est maintenant terminé.

9 ROLES

En s'inspirant du chapitre 7 sur les rôles, il va falloir scinder en 2 le playbook installant et configurant Apache sur RHEL et Debian.

On doit donc obtenir un playbook principal, qui en fonction d'une condition sur le type d'OS détecté, va faire appel à un rôle Debian ou un rôle RHEL pour installer Apache sur le système cible.

Une façon de charger un rôle spécifique consiste à faire appel au module ansible.builtin.include_role, par exemple les lignes ci-dessous vont chercher par défaut la fichier tasks/main.yml du rôle apache-debian :

```
      ansible.builtin.include_role:
        name: apache-debian
```

10 GESTION DE SECRETS

Dans cet exercice, on chiffre des variables avec Ansible Vault, et on lance un playbook pour les utiliser.

Créer un fichier `secret_data.yml`, tel que :

```
---
http_admin_passwd: H33_%passw0rd
psql_admin_passwd: p0stgreS@l-i4
```

Procéder au chiffrement de ce fichier avec la commande '`ansible-vault`', en utilisant '`hello`' comme clef de chiffrement :

```
$ ansible-vault encrypt secret_data.yml
New Vault password:
Confirm New Vault password:
Encryption successful
```

Visualiser ce fichier avec la commande ansible-vault view :

```
$ ansible-vault view secret_data.yml
Vault password:
---
http_admin_passwd: H33_%passw0rd
psql_admin_passwd: p0stgreS@l-i4
```

Créer un playbook nommé '`display_data.yml`'. Ce playbook doit contenir deux tâches `ansible.builtin.debug`, pour afficher chacune des deux variables (l'ordre n'a pas d'importance).

L'exercice est maintenant terminé.

11 IDENTIFIER LA VERSION DE PYTHON

Lorsque les facts sont collectés sur une cible, le chemin de l'interpréteur Python est contenu dans une variable `ansible_facts['discovered_interpreter_python']`.

Utiliser le module '`command`' pour exécuter la commande "`interpréteur_python --version`", et capturer le résultat dans une variable '`python_version`'. Afficher ensuite l'identifiant de version ainsi découvert (il faut afficher le contenu de la variable 'python_version.stdout').

En utilisant le plugin de test 'version' (https://docs.ansible.com/ansible/latest/collections/ansible/builtin/version_test.html), afficher un message indiquant si on a affaire à une version supportée ou non de Python.

Exemple de playbook :

```
---
- hosts: all
  gather_facts: yes

  tasks:

    - ansible.builtin.command:
        cmd: "{{ ansible_facts['discovered_interpreter_python'] }} --version"
      register: output

    - ansible.builtin.set_fact:
        python_version: "{{ output.stdout | split(' ') }}"


    - ansible.builtin.debug:
        msg: "{{ inventory_hostname }} : version non supportée ({{
python_version[1] }})"
      when: python_version[1] is version('3.7','<=')

    - ansible.builtin.debug:
        msg: "{{ inventory_hostname }} : version supportée ({{ python_version[1]
}})"
      when: python_version[1] is version('3.7','>')
```

L'exercice est maintenant terminé.

12 AMÉLIORER UN MESSAGE D'ERREUR

Écrire le playbook '`erreur1.yml`' ci-dessous :

```
---
- hosts: localhost
  gather_facts: no

  tasks:
    - ansible.builtin.copy:
        src: monfichier
        dest: /tmp/monfichier2
```

S'assurer qu'il n'existe aucun fichier de nom '`monfichier`' dans le répertoire en cours, ou dans un sous-répertoire '`files`'.

Lancer le playbook :

```
$ ansible-playbook erreur1.yml
```

Celui affiche un message d'erreur similaire à celui reproduit ci-dessous :

```
[WARNING]: provided hosts list is empty, only localhost is available. Note that
the implicit localhost does not match 'all'

PLAY [localhost]
*******************************************************************************
*******************************************************************

TASK [ansible.builtin.copy]
*******************************************************************************
*************************************************************
An exception occurred during task execution. To see the full traceback, use -vvv.
The error was: If you are using a module and expect the file to exist on the
remote, see the remote_src option
fatal: [localhost]: FAILED! => {"changed": false, "msg": "Could not find or access
'monfichier'\nSearched
in:\n\t/home/admjkl/ansible/tmp/files/monfichier\n\t/home/admjkl/ansible/tmp/monfi
chier\n\t/home/admjkl/ansible/tmp/files/monfichier\n\t/home/admjkl/ansible/tmp/mon
fichier on the Ansible Controller.\nIf you are using a module and expect the file
to exist on the remote, see the remote_src option"}

PLAY RECAP
*******************************************************************************
*******************************************************************************
localhost                  : ok=0    changed=0    unreachable=0    failed=1
skipped=0    rescued=0    ignored=0
```

Relancer le playbook, mais cette fois avec l'option -vvv.

```
$ ansible-playbook erreur1.yml -vvv
```

Le message d'erreur est formaté de façon plus lisible, bien que noyé au milieu d'une multitude d'informations :

```
[... blah blah blah ...]
The full traceback is:
Traceback (most recent call last):
  File "/usr/lib/python3.9/site-packages/ansible/plugins/action/copy.py", line
466, in run
    source = self._find_needle('files', source)
  File "/usr/lib/python3.9/site-packages/ansible/plugins/action/__init__.py", line
1431, in _find_needle
    return self._loader.path_dwim_relative_stack(path_stack, dirname, needle)
  File "/usr/lib/python3.9/site-packages/ansible/parsing/dataloader.py", line 341,
in path_dwim_relative_stack
    raise AnsibleFileNotFound(file_name=source, paths=[to_native(p) for p in
search])
ansible.errors.AnsibleFileNotFound: Could not find or access 'monfichier'
Searched in:
        /home/admjkl/ansible/tmp/files/monfichier
        /home/admjkl/ansible/tmp/monfichier
        /home/admjkl/ansible/tmp/files/monfichier
        /home/admjkl/ansible/tmp/monfichier on the Ansible Controller.
If you are using a module and expect the file to exist on the remote, see the
remote_src option
fatal: [localhost]: FAILED! => {
    "changed": false,
    "invocation": {
        "dest": "/tmp/monfichier2",
        "module_args": {
            "dest": "/tmp/monfichier2",
            "src": "monfichier"
        },
        "src": "monfichier"
    },
    "msg": "Could not find or access 'monfichier'\nSearched
in:\n\t/home/admjkl/ansible/tmp/files/monfichier\n\t/home/admjkl/ansible/tmp/monfi
chier\n\t/home/admjkl/ansible/tmp/files/monfichier\n\t/home/admjkl/ansible/tmp/mon
fichier on the Ansible Controller.\nIf you are using a module and expect the file
to exist on the remote, see the remote_src option"
}

PLAY RECAP
*********************************************************************************
*********************************************************************************
localhost                  : ok=0    changed=0    unreachable=0    failed=1
skipped=0    rescued=0    ignored=0
```

Bien que plus lisible, le message reste néanmoins très agressif.

Éditer le fichier '`ansible.cfg`' du répertoire de travail. Dans la section `[defaults]`, rajouter la ligne suivante :

```
stdout_callback = yaml
```

Relancer le playbook, SANS le `-vvv`, comme ci-dessous :

```
$ ansible-playbook erreur1.yml
```

On obtient un affichage similaire à celui-ci-dessous :

```
[WARNING]: provided hosts list is empty, only localhost is available. Note that
the implicit localhost does not match 'all'

PLAY [localhost]
*********************************************************************************
*********************************************************************

TASK [ansible.builtin.copy]
*********************************************************************************
*********************************************************************
An exception occurred during task execution. To see the full traceback, use -vvv.
The error was: If you are using a module and expect the file to exist on the
remote, see the remote_src option
fatal: [localhost]: FAILED! => changed=false
  msg: |-
    Could not find or access 'monfichier'
    Searched in:
            /home/admjkl/ansible/tmp/files/monfichier
            /home/admjkl/ansible/tmp/monfichier
            /home/admjkl/ansible/tmp/files/monfichier
            /home/admjkl/ansible/tmp/monfichier on the Ansible Controller.
    If you are using a module and expect the file to exist on the remote, see the
remote_src option

PLAY RECAP
*********************************************************************************
*********************************************************************
localhost                  : ok=0    changed=0    unreachable=0    failed=1
skipped=0    rescued=0    [WARNING]: provided hosts list is empty, only localhost
is available. Note that the implicit localhost does not match 'all'
```

Certes l'erreur n'est pas (encore) réglée, mais elle est déjà plus compréhensible.

Commentaire : on a en fait configuré Ansible pour utiliser un module d'affichage spécifique, ici celui qui formate la sortie en yaml, et qui porte justement ce nom. Ces modules d'affichages sont appelés '`callback plugin`'; il en existe certains qui permettent par exemple d'afficher le temps passé dans chaque tâche, ainsi qu'un résumé d'exécution.

FIN des exercices 'Ansible pour professionnel Unix/Linux'
