Chapitre 4
Playbooks

1. Définition

Il peut s'avérer qu'exécuter une à une les différentes commandes ad hoc devienne assez rapidement rébarbatif. D'autant plus qu'elles exécutent une tâche simple ponctuellement sur un ou plusieurs hôtes gérés.

S'il faut lancer des tâches plus élaborées sur des hôtes et pouvoir les reproduire facilement, il est alors possible d'utiliser les playbooks, lesquels représentent un ensemble de scripts d'automatisation que l'on appelle « plays ».

Les playbooks définissent, donc, les tâches de gestion de configuration à accomplir sur les hôtes gérés.

2. Écriture d'un playbook

Vous pouvez concevoir des playbooks de bout en bout ou à partir de modules disponibles au sein de la plateforme, voire depuis la communauté d'utilisateurs.

Le format des fichiers playbooks est YAML. Nous avons déjà rencontré ce langage dans le chapitre Présentation de Ansible pour la rédaction des inventaires. Vous allez de nouveau utiliser l'éditeur de texte vi avec des paramètres spécifiques à YAML dans le fichier de configuration *~/.vimrc*.

Exemple

Avant de commencer, créez un répertoire de travail nommé **workspace** :

```
root@server1 ~]# mkdir workspace ; cd workspace
[root@server1 workspace]#
```

Sur les hôtes *server2* et *server3*, qui sont des plateformes Linux, créez un utilisateur avec les caractéristiques suivantes :

Nom : john

Groupe primaire : users

Groupes supplémentaires : adm et sys.

Vous pouvez appliquer la commande ad hoc ci-dessous pour effectuer cette tâche :

```
[root@server1 workspace]# ansible -m user -a "name=john \
> group=users groups=adm,sys state=present" all
[root@server1 workspace]#
```

Le module user de Ansible gère les comptes utilisateurs des plateformes Linux, macOS et Unix. Le module à utiliser, quant aux plateformes Microsoft Windows, est *win_user*.

Passez des arguments au module user avec le commutateur `-a` :

Paramètres	Valeurs
`name`	Nom de l'utilisateur à créer, supprimer ou modifier.
`group`	Nom du groupe primaire.
`groups`	Noms des groupes supplémentaires.
`state`	Si le compte doit exister ou non, prendre des mesures si l'état est différent de ce qui est indiqué. La valeur peut être « `present` » ou « `absent` ».

Le mot `all` définit que l'action est destinée à tous les hôtes de l'inventaire.

Cette commande peut être aussi écrite en tant que « play » dans un fichier playbook que l'on va nommer *user_john.yml* :

```
[root@server1 workspace]# cat user_john.yml
---
- name: Création de l'utilisateur john
  hosts: all
  tasks:
    - name: john Doe
      user:
        name: john
        group: users
        groups: adm,sys
        state: present
[root@server1 workspace]#
```

La balise `name` spécifie le nom du playbook Ansible. N'importe quel nom logique peut être donné.

La balise `hosts` spécifie la liste des hôtes ou le groupe d'hôtes à gérer.

La balise `vars` permet de définir les variables que vous souhaitez utiliser dans le playbook. Leur utilisation est similaire aux variables rencontrées dans n'importe quel langage de programmation.

La balise `tasks` décrit la liste d'actions à effectuer. Tous les playbooks doivent contenir des tâches ou une liste de tâches à exécuter. Chaque tâche est liée à un module Ansible, qui est exécuté avec les arguments.

3. Exécution de playbooks

Sur le nœud de contrôle, la commande `ansible-playbook` exécute les playbooks. Il faut lui fournir en tant qu'argument le nom du playbook à utiliser.

3.1 Principe de fonctionnement

La définition des attributs `name` dans un playbook qui contient plusieurs plays et tâches permet de surveiller la progression de leur exécution.

Gathering Facts est une tâche particulière qui, généralement, est exécutée automatiquement par le module *setup* au début d'un play. Ce sujet est abordé dans le chapitre Gestion des faits.

Les plays et les tâches sont traités dans l'ordre dans lequel ils sont écrits au sein du playbook. Lors de leur exécution, la sortie générée indique les résultats de chaque tâche effectuée.

Exécution du playbook user_john.yml

```
[root@server1 workspace]# ansible-playbook user_john.yml

PLAY [Création de l'utilisateur john]*****************************

TASK [Gathering Facts]*******************************************
ok: [172.16.32.1]
ok: [172.16.32.2]

TASK [john Doe]**************************************************
changed: [172.16.32.1]
changed: [172.16.32.2]

PLAY RECAP*******************************************************
172.16.32.1   : ok=2  changed=1  unreachable=0  failed=0  skipped=0
rescued=0  ignored=0
172.16.32.2   : ok=2  changed=1  unreachable=0  failed=0  skipped=0
rescued=0  ignored=0

[root@server1 workspace]#
```

Vous pouvez, sans crainte, exécuter un playbook plusieurs fois du fait que les tâches au sein de celui-ci sont idempotentes.

3.2 Verbosité des playbooks

La commande `ansible-playbook` fournit, par défaut, peu d'informations sur l'exécution d'une tâche. Pour obtenir un retour plus détaillé, il existe quatre niveaux de verbosité :

Commutateurs	Description
`-v`	Afficher les résultats de la tâche.
`-vv`	Afficher les résultats et la configuration de la tâche.
`-vvv`	Afficher les résultats et la configuration de la tâche et aussi les informations sur les connexions aux hôtes cibles.
`-vvvv`	Idem que `-vvv` avec des informations détaillées supplémentaires telles que l'activation du débogage de connexion.

3.3 Vérification de la syntaxe

Vous disposez de deux outils pour effectuer la vérification syntaxique :

– `ansible-playbook` avec le commutateur `--syntax-check` ;

– `yamllint`.

Remarque

La commande qu'il est conseillé d'utiliser est `ansible-playbook`. Elle est, de fait, disponible sur le nœud de contrôle tandis que `yamllint` doit être installée depuis l'EPEL comme nous l'avons vu dans le chapitre Déploiement.

3.3.1 Vérification avec Ansible-playbook

Il est préférable de réaliser une vérification syntaxique avant d'exécuter un playbook et de s'assurer ainsi que son contenu est correct. Il faut, pour cela, utiliser le commutateur `--syntax-check` de la commande `ansible-playbook` :

```
[root@server1 workspace]# ansible-playbook --syntax-check \
> user_john.yml -vv
ansible-playbook 2.8.5
 config file = /root/playbook/ansible.cfg
 configured module search path = ['/root/.ansible/plugins/modules',
'/usr/share/ansible/plugins/modules']
 ansible python module location = /usr/lib/python3.6/
site-packages/ansible
 executable location = /usr/bin/ansible-playbook
 python version = 3.6.8 (default, Oct  7 2019, 17:58:22)
[GCC 8.2.1 20180905 (Red Hat 8.2.1-3)]
Using /root/playbook/ansible.cfg as config file
1 plays in user_john.yml

playbook: user_john.yml
[root@server1 workspace]#
```

Il est aussi possible de combiner la vérification syntaxique et la verbosité comme vous pouvez le constater.

La détection d'une erreur de syntaxe se représente de cette façon :

```
[root@server1 workspace]# ansible-playbook --syntax-check \
> user_john.yml -vv
ansible-playbook 2.8.5
 config file = /root/playbook/ansible.cfg
 configured module search path = ['/root/.ansible/plugins/modules',
'/usr/share/ansible/plugins/modules']
 ansible python module location = /usr/lib/python3.6/
site-packages/ansible
 executable location = /usr/bin/ansible-playbook
 python version = 3.6.8 (default, Oct  7 2019, 17:58:22)
[GCC 8.2.1 20180905 (Red Hat 8.2.1-3)]
Using /root/playbook/ansible.cfg as config file
ERROR! Syntax Error while loading YAML.
 could not find expected ':'

The error appears to be in '/root/playbook/user_john.yml':
line 7, column 13,
```

```
but may be elsewhere in the file depending on the exact syntax problem.

The offending line appears to be:

     user
       name: john
           ^ here

[root@server1 workspace]#
```

L'erreur se situe approximativement sur la ligne 7 qui est pointée par un curseur « ^ *here* ». Généralement, elle se situe sur une ligne située avant. Pour vous en assurer, consultez le contenu du fichier *user_john.yml* :

```
[root@server1 workspace]# cat -n user_john.yml
    1        ---
    2        - name: Création de l'utilisateur john
    3         hosts: all
    4         tasks:
    5           - name: john Doe
    6             user
    7               name: john
    8               group: users
    9               groups: adm,sys
   10               state: present
[root@server1 workspace]#
```

L'erreur est en réalité sur la ligne 6. Un caractère « `:` » est absent derrière le mot-clé `user`.

3.3.2 Vérification avec yamllint

En reprenant le même fichier erroné que précédemment :

```
[root@server1 workspace]# yamllint user_john.yml -vv
user_john.yml
 7:13      error    syntax error: could not find expected ':' (syntax)

[root@server1 workspace]#
```

L'erreur se situe sur la ligne 7, colonne 13, c'est-à-dire sur le caractère « `:` » situé après le mot-clé `name` : `Yamllint` s'est arrêté également au même endroit que la commande `ansible-playbook`.