Ansible Task Control

1. Boucles

Tous les langages de programmation fournissent un moyen d'itérer sur les données pour effectuer des tâches répétitives. Ansible fournit également un moyen de faire de même en utilisant un concept appelé boucle , qui est fourni par les plugins de recherche Ansible. Avec les boucles, une seule tâche dans un playbook peut être utilisée pour créer plusieurs utilisateurs, installer de nombreux packages, etc.

Bien qu'il existe de nombreuses façons d'utiliser les boucles dans Ansible, nous n'en couvrirons qu'une seule pour vous aider à démarrer. La façon la plus simple d'utiliser des boucles dans ansible est d'utiliser un mot-clé with_items, qui est utilisé pour parcourir une liste d'éléments pour effectuer certaines tâches répétitives. Le playbook suivant comprend une tâche qui installe les packages en boucle à l'aide du mot-clé with_items.

```
---
- hosts: hosts_nodes
  gather_facts: yes

  tasks:

  - name: Installing packages using loops
    yum: name={{ item }} state=present
    with_items:
      - nmap
      - htop
      - tree
```

Ou sous la forma suivante :

```
---
- hosts: hosts_nodes
  gather_facts: yes
  vars:
    packages:
      - nmap
      - htop
      - tree

  tasks:

  - name: Installing packages using loops
    yum: name={{ item }} state=present
    with_items: {{ packages }}
```

2. Les balises Tags

Les *tags* vous permettent d'exécuter uniquement des tâches spécifiques à partir de votre playbook via la ligne de commande. Ajoutez simplement le mot clé *tags* - pour chaque tâche et exécutez uniquement la ou les tâches que vous souhaitez en utilisant l'option *--tags* à la fin de la commande *ansible*. Dans le playbook suivant, nous avons ajouté des balises à la fin de chaque tâche, ce qui nous permet d'exécuter des tâches séparément d'un seul playbook.

Le playbook suivant contient deux tâches; la première est taguée avec la balise webserver, l'autre n'a aucune balise associée:

```
[ansible@master01 base]$ vim tags_playbook.yml
---
- name: Example play using tagging
  hosts: host01

  tasks:
    - name: httpd is installed
      yum:
        name: httpd
        state: latest
      tags: webserver
    - name: postfix is installed
      yum:
        name: postfix
        state: present
```

Pour exécuter uniquement la première tâche, l'argument --tags peut être utilisé:

```
[ansible@master01 base]$ansible-playbook --tags webserver tags_playbook.yml

PLAY [Example play using tagging]
**********************************************************************

TASK [Gathering Facts]
**********************************************************************
ok: [host01]

PLAY RECAP
**********************************************************************
host01   : ok=1  changed=0  unreachable=0  failed=0  skipped=0  rescued=0
ignored=0
```

Étant donné que l'option --tags a été spécifiée, le playbook a uniquement exécuté la tâche balisée avec la balise *webserver*. Pour ignorer les tâches avec une balise spécifique et exécuter uniquement les tâches sans cette balise, l'option --skip-tags peut être utilisée:

```
[master01 base]$ ansible-playbook --skip-tags webserver tags_playbook.yml

PLAY [Example play using tagging]
**********************************************************************

TASK [Gathering Facts]
**********************************************************************
ok: [host01]
```

```
TASK [postfix is installed]
*********************************************************************
ok: [host01]

PLAY RECAP
*********************************************************************
host01  : ok=2  changed=0  unreachable=0  failed=0  skipped=0  rescued=0
ignored=0
```

3. Exécution conditionnelle des tâches

Ansible peut utiliser des conditions pour exécuter des tâches ou des Plays lorsque certaines conditions sont remplies. Par exemple, un conditionnel peut être utilisé pour déterminer si le service Apache est démarré avant d'interroger une URL. Nous allons utiliser les ansible_facts :

```
[ansible@master01 base]$ vim when_playbook.yml
---
- hosts: host01
  tasks:
  - name: populate service facts
    service_facts:

  - debug:
      var: ansible_facts.services["httpd.service"].state
  - name: Check that you can connect (GET) to host01
    uri:
      url: http://host01
    when: ansible_facts.services["httpd.service"].state == "running"

[ansible@master01 base]$ ansible-playbook when_playbook.yml
```

L'exemple suivant montre deux tâches conditionnelles basées sur le résultat d'une commande *php -v*. Nous allons tester le statut de sortie de la commande, car nous savons que son exécution échouera si PHP n'est pas installé sur ce serveur. La partie *ignore_errors* de la tâche est importante pour s'assurer que le provisioning continue même lorsque l'exécution de la commande échoue.

```
[ansible@master01 base]$ vim php_playbook.yml
---
- hosts: host01
  tasks:
  - name: Check if PHP is installed
    command: php -v
    register: php_installed
    ignore_errors: true

  - name: This task is only executed if PHP is installed
    debug: var=php_installed
    when: php_installed.rc == 0

  - name: This task is only executed if PHP is NOT installed
    debug: msg='PHP is NOT installed'
    when: php_installed.rc != 0

[ansible@master01 base]$ ansible-playbook php_playbook.yml
```

```
PLAY [host01] ******************************************************************

TASK [Gathering Facts]
*********************************************************************************
ok: [host01]

TASK [Check if PHP is installed]
*********************************************************************************
fatal: [host01]: FAILED! => {"changed": false, "cmd": "php -v", "msg":
"[Errno 2] Aucun fichier ou dossier de ce type", "rc": 2}
...ignoring

TASK [This task is only executed if PHP is installed]
*********************************************************************************
skipping: [host01]

TASK [This task is only executed if PHP is NOT installed]
*********************************************************************************
ok: [host01] => {
    "msg": "PHP is NOT installed"
}

PLAY RECAP
*********************************************************************************
host01       : ok=3    changed=0    unreachable=0    failed=0    skipped=1
rescued=0    ignored=1
```

4. Travailler avec des modèles (templates)

Les modèles sont généralement utilisés pour régler les fichiers de configuration, ce qui permet d'utiliser des variables et d'autres fonctionnalités destinées à rendre ces fichiers plus polyvalents et réutilisables.

En règle générale, après avoir installé un serveur Web comme Apache, vous devez configurer un fichier d'hôtes virtuels pour servir correctement un site Web donné sur votre VPS. Au lieu d'utiliser SSH pour vous connecter à votre VPS pour le configurer après avoir exécuté Ansible, ou d'utiliser le module *copy* d'Ansible pour copier de nombreux fichiers de configuration uniques individuellement, vous pouvez profiter des fonctionnalités des *Templates* d'Ansible .

Un fichier *template* contient tous les paramètres de configuration dont vous avez besoin, tels que les paramètres d'hôte virtuel Apache, et utilise des variables, qui sont remplacées par les valeurs appropriées lorsque le playbook est exécuté. Les fichiers de modèle se terminent généralement par l'extension *.j2* qui désigne le moteur de création de modèles *Jinja2*.

Pour commencer à travailler avec des modèles, créez un répertoire pour les fichiers de modèles dans votre répertoire de travail Ansible.

```
[ansible@master01 base]$ mkdir templates
L'exemple suivant est un modèle de configuration d'un hôte virtuel Apache,
utilisant une variable pour configurer la racine du document pour cet
hôte :
[ansible@master01 base]$ vim vars/sitesvars.yml
doc_root:
  - site1
  - site2
```

```
[ansible@master01 base]$ vim templates/vhosts.j2
<VirtualHost *:80>
   ServerAdmin webmaster@localhost
   DocumentRoot {{ doc_root }}

   <Directory {{ doc_root }}>
       AllowOverride All
       Require all granted
   </Directory>
</VirtualHost>
```

```
[ansible@master01 base]$ vim templates_playbook.yml
---
- name: Vhosts Templale Example
  hosts: host01
  tasks:
  - name: Includes doc_root Varaibles for Apache
    include_vars: vars/sitesvars.yml

  - name: Change default Apache virtual host
    template:
      src: vhosts.j2
      dest: /etc/httpd/conf.d/{{ item }}
    with_items: "{{ doc_root }}"
```

```
[ansible@master01 base]$ ansible-playbook templates_playbook.yml

PLAY [Vhosts Templale Example]
*********************************************************************

TASK [Gathering Facts]
*********************************************************************
ok: [host01]

TASK [Includes doc_root Varaibles for Apache]
*********************************************************************
ok: [host01]

TASK [Change default Apache virtual host]
*********************************************************************
changed: [host01] => (item=site1)
changed: [host01] => (item=site2)

PLAY RECAP
*********************************************************************
host01      : ok=3    changed=1    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
```