Ansible Playbook

1. Présentation

L'élément central d'Ansible est le Playbook. C'est un script de gestion de configuration : on lui indique les machines à configurer et les opérations à exécuter sur ces hôtes.

Un playbook est un fichier texte écrit au format YAML, et est normalement enregistré avec l'extension yml.

Le playbook utilise principalement l'indentation avec des caractères d'espace pour indiquer la structure de ses données. YAML n'impose pas d'exigences strictes sur le nombre d'espaces utilisés pour l'indentation et il est souhaitable de ne pas utiliser les tabulations.

Pour vous aider à comprendre le format d'un playbook, nous utiliserons une commande ad hoc simple, par exemple, la commande suivante utilise le module *user* pour s'assurer que l'utilisateur *newbie* existe et possède l'UID 4000 sur host01 (il sera créé sinon) :

```
$ ansible -m user -a 'name=newbie uid=4000 state=present' hosts_nodes
```

Cela peut être réécrit comme une tâche unique simple et enregistré dans un playbook :

```
$ vim newbie_playbook.yml
---
- name: Add newbie user to hosts_nodes group
  hosts: hosts_nodes
  tasks:
    - name: newbie exists with UID 4000
      user:
        name: newbie
        uid: 4000
        state: present
```

2. Vérification de la syntaxe

Avant d'exécuter un Playbook, il est recommandé d'effectuer une vérification pour s'assurer que la syntaxe de son contenu est correcte. La commande *ansible-playbook --syntax-check* peut être utilisée pour vérifier la syntaxe d'un fichier playbook. L'exemple suivant montre la vérification de la syntaxe réussie d'un playbook.

```
[ansible@master01 base]$ ansible-playbook --syntax-check newbie_playbook.yml

playbook: newbie_playbook.yml
```

Une autre option utile est l'option -C. Elle permet à Ansible de signaler les modifications qui se seraient produites si le playbook avait été exécuté, mais n'apporte aucune modification réelle aux hôtes gérés.

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

Pour exécuter ce Playbook on utilise la commande suivante :

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

3. Playbook Nginx

Nous allons découvrir un exemple d'utilisation plus élaboré nous installerons puis configurerons un serveur *nginx*. Plaçons nous dans le scénario où l'utilisateur *ansible* souhaite configurer le serveur web *nginx*. Dans le répertoire *files*, créons un fichier appelé *nginx_playbook.yml* qui correspondra au Playbook d'installation et de configuration de *nginx*.

Voici l'arborescence du projet :

```
[ansible@master01 base]$ tree
.
├── ansible.cfg
├── base.log
├── files
│   ├── index.html
│   ├── nginx.conf
│   └── nginx_playbook.yml
└── inventory.ini
```

Le contenu des fichiers de configuration de *nginx* sera le suivant :

files/index.html

```
<!doctype html>
<html>
  <head>
    <title>Nginx Playbook!
</title>
  </head>
  <body>
    <p><strong>This is a sample
index.html on Nginx server
deployed with Ansible Playbook
</strong></p>
  </body>
</html>
```

files/nginx.conf

```
user nginx;
include /usr/share/nginx/modules/*.conf;
events {
    worker_connections 1024;
}
http {
  include /etc/nginx/conf.d/*.conf;

  server {
    listen       80 default_server;
    server_name  localhost;
    root     /usr/share/nginx/html;
    include /etc/nginx/default.d/*.conf;
        location / {
        }
        error_page 404 /404.html;
            location = /40x.html {
        }
    }
}
```

Le contenu du Playbook *nginx_playbook.yml* :

```
[ansible@master01 base]$ vim files/nginx_playbook.yml
---
- name: Configure webserver with nginx
  hosts: host02
  become: true

  tasks:
    - name: Install nginx
      yum:
        name: "nginx"
```

```
        update_cache: no
        state: present

    - name: copy nginx config file
      copy:
        src: 'files/nginx.conf'
        dest: '/etc/nginx/'
        force: yes

    - name: copy index.html file
      copy:
        src: 'files/index.html'
        dest: '/usr/share/nginx/html'
        force: yes

    - name: restart nginx
      systemd:
        name: nginx
        state: restarted
        enabled: yes
...
```

Exécutez le Playbook :

```
[ansible@master01 base]$ ansible-playbook files/nginx_playbook.yml
```

Vérifiez que Nginx est présent :

```
[ansible@master01 base]$ curl host02
<!doctype html>
<html>
  <head>
    <title>Nginx Playbook !</title>
  </head>
  <body>
    <p><strong>This is a sample index.html on Nginx server deployed with
Ansible Playbook </strong></p>
  </body>
</html>
```

Désinstallez Nginx de host02

```
[ansible@master01 base]$ ansible host02 -m yum -a "name=nginx state=absent"
```

4. Playbook avec plusieurs plays

Un playbook est un fichier YAML contenant une liste d'une ou plusieurs lectures (play). Un play unique est une liste ordonnée de tâches (tasks) à exécuter contre des hôtes sélectionnés dans l'inventaire. Par conséquent, si un playbook contient plusieurs plays, chaque play peut appliquer ses tâches à un ensemble d'hôtes distinct.

Cela peut être très utile lors de l'orchestration d'un déploiement complexe qui peut impliquer différentes tâches sur différents hôtes.

L'exemple suivant montre un Playbook simple avec deux jeux. La première lecture s'exécute sur host01 pour installer Apache et la deuxième lecture s'exécute sur host02 pour installer Mariadb.

La rédaction d'un livre de jeu contenant plusieurs pièces est très simple. Chaque jeu dans le playbook est écrit comme un élément de liste de niveau supérieur dans le playbook. Chaque jeu est un élément de liste contenant les directives de jeu habituelles.

```
[ansible@master01 base]$ vim multiplay.yml
---
# This is a simple playbook with two plays
- name: first play
  hosts: host01
  tasks:
    - name: first task
      yum:
        name: httpd
        state: present
    - name: second task
      systemd:
        name: httpd
        enabled: true
- name: second play
  hosts: host02
  tasks:
    - name: first task
      yum:
        name: mariadb-server
        state: present
    - name: second task
      systemd:
        name: mariadb
        enabled: true
```

5. Les variables

Les variables permettent une entrée dynamique. Dans Ansible, les variables peuvent être définies de plusieurs manières telles que: playbook lui-même, fichier d'inventaire via l'hôte ou le groupe vars, cli via extra_vars ou dans un fichier vars séparé qui peut être inclus dans le playbook. Jetons un coup d'œil à ces options plus en détail. Au lieu de coder en dur le nom d'utilisateur que nous avons créé gans le premier Playbook, explorons comment le faire en utilisant les différentes options de variable.

Les variables sont applées en format Jinja2 sous la forme : {{ interface }} ou encore une autre variable {{ ipv4.address }}.

Les guillemets sont nécessaires dans l’appel aux variables dans les paramètres car on appelle sa valeur. Par contre dans les opérations logiques telles que les conditions cette mise en guillemets n’est pas nécessaire.

5.1 Vars via le fichier d'inventaire.

Des variables peuvent être définies pour un hôte ou un groupe dans le fichier d'inventaire lui-même. Ici, nous allons définir un nom d'utilisateur variable pour le groupe *hosts_nodes*.

```
[ansible@master01 base]$ vim inventory.ini
[masters_nodes]
master01

[hosts_nodes]
host0[1:3] ansible_user=ansible

[hosts_nodes:vars]
username=newbie

[lab_nodes:children]
masters_nodes
hosts_nodes
```

Mettre à jour le Playbook *newbie_playbook.yml* pour utiliser des variables.

```
[ansible@master01 base]$ vim newbie_playbook2.yml
---
- name: Configure important user consistently
  hosts: hosts_nodes
  tasks:
    - name: newbie exists with UID 4000
      user:
        name: "{{ username }}"
        uid: 4000
        state: present

    - name: Add message to /etc/issue file
      lineinfile:
        dest: /tmp/{{ username }}
        line: "{{ username }} user added"
        create: yes
        state: present
```

Exécutez le playbook:

```
[ansible@master01 base]$ ansible-playbook newbie_playbook2.yml
```

5.2 Variables dans le playbook

Les variables peuvent être définies directement dans le playbook.

```
[ansible@master01 base]$ vim newbie_playbook3.yml
---
- name: Configure important user consistently
  hosts: hosts_nodes
  vars:
    username: newbie

  tasks:
    - name: newbie exists with UID 4000
      user:
        name: "{{ username }}"
```

```
        uid: 4000
        state: present

    - name: Add message to /etc/issue file
      lineinfile:
        dest: /etc/issue
        line: "{{ username }} user added"
        create: yes
        state: present
```

5.3 Variables importées du fichier vars.

Semblable à l'exemple ci-dessus, les variables peuvent être définies dans un fichier séparé puis importées dans le playbook.

Créez un fichier *my_vars.yml* :

```
[ansible@master01 base]$ vim my_vars.yml
---
username: newbie
```

Mettre à jour le playbook.

```
[ansible@master01 base]$ vim newbie_playbook4.yml
---
- name: Configure important user consistently
  hosts: hosts_nodes
  vars_files:
     - ./my_vars.yml

  tasks:
    - name: newbie exists with UID 4000
      user:
        name: "{{ username }}"
        uid: 4000
        state: present

    - name: Add message to /etc/issue file
      lineinfile:
        dest: /etc/issue
        line: "{{ username }} user added"
        create: yes
        state: present
```

La meilleure pratique consiste à utiliser le fichier d'inventaire pour définir les variables de type clé/valeur qui doivent être paramétrées. Les variables générées dynamiquement ou utilisant des structures de données imbriquées doivent utiliser vars_files et être incluses. Si possible, évitez d'utiliser directement les variables dans le playbook.

6. Ansible Facts

Chaque fois qu'Ansible est exécuté, sauf s'il est désactivé, le module *setup* est également exécuté. Le module *setup* rassemble des *facts* Ansible. Ce sont des variables qui nous donnent des informations précieuses sur l'hôte géré et elles peuvent être traitées dans les playbooks. Tout ce qui provient d'un réseau d'hôtes, du matériel et des informations sur le système d'exploitation est collecté. Il est également possible de définir des *facts* personnalisés qui seraient collectés.

6.1 Afficher les facts Ansible pour un hôte.

On a déjà vu dans un Lab précédent la commande Ad hoc pour afficher les *facts* :

```
[ansible@master01 base]$ ansible -m setup host01
host01 | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "172.20.10.193",
            "10.20.30.11"
        ],
        "ansible_all_ipv6_addresses": [],
        "ansible_apparmor": {
            "status": "disabled"
        },
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "07/29/2019",
        "ansible_bios_version": "6.00",
...
```

6.2 Utilisez des facts ansibles dans le Playbook

Ici, nous allons imprimer la mémoire et le nombre de cœurs de processeur dans notre playbook en ajoutant une nouvelle tâche.

```
[ansible@master01 base]$ vim newbie_playbook.yml
---
- name: Configure important user consistently
  hosts: hosts_nodes
  vars_files:
     - ./my_vars.yml

  tasks:
    - name: Print Memory and CPU Cores
      debug:
        msg: >
          "Host {{ ansible_hostname }}
          has {{ ansible_memtotal_mb }} MB Memory
          and {{ ansible_processor_cores }} CPU Cores."

    - name: newbie exists with UID 4000
      user:
        name: "{{ username }}"
        uid: 4000
        state: present

    - name: Add message to /etc/issue file
      lineinfile:
        dest: /etc/issue
        line: "{{ username }} user added"
        create: yes
        state: present
```

Exécutez le Playbook :

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

```
PLAY [Configure important user consistently]
*********************************************************************

TASK [Gathering Facts]
*********************************************************************
ok: [host02]
ok: [host03]
ok: [host01]

TASK [Print Memory and CPU Cores]
*********************************************************************
ok: [host01] => {
    "msg": "\"Host host01  has 972 MB Memory  and 1 CPU Cores.\"\n"
}
ok: [host02] => {
    "msg": "\"Host host02  has 972 MB Memory  and 1 CPU Cores.\"\n"
}
ok: [host03] => {
    "msg": "\"Host host03  has 972 MB Memory  and 1 CPU Cores.\"\n"
}
…
```

6.3 Les handlers, include, debug

Un *handler* est identique à une tâche (task), mais il sera exécuté lorsqu'il sera appelé par une autre tâche. C'est comme un système événementiel. Un *handler* exécutera une tâche uniquement lorsqu'elle est appelée par un événement qu'il écoute.

Les tâches peuvent être incluses dans un playbook à partir d'un fichier externe à l'aide de la directive *include*.

```
tasks:
- name: Include tasks to install the database server
  include: tasks/db_server.yml
```

Le module include_vars peut inclure des variables définies dans des fichiers JSON ou YAML, remplaçant les variables hôtes et les variables playbook déjà définies.

```
tasks:
- name: Include the variables from a YAML or JSON file
  include_vars: vars/variables.yml
```

Le module *debug* fournit des informations de débogage supplémentaires lors de l'exécution d'un playbook (par exemple, la valeur actuelle d'une variable), il peut servir aussi pour afficher des messages.

Dans l'exemple suivant nous allons déployer Apache sur les trois hôtes, en un premier temps nous allons créer les fichiers de variables *vars/apachevars.yml* et *vars/*`mariadbvars.yml` contenant trois variable : package, service et state. Puis un fichier de tâches *tasks/instruntask.yml* contenant deux tâches l'une pour installer le service et l'autre pour le démarrer. Enfin notre playbook avec une tache handler pour le redémarrage des services quand cela est nécessaire.

L'arborescence du projet est la suivante :

```
[ansible@master01 base]$ tree
```

```
.
├── ansible.cfg
├── base.log
├── multiplay2.yml
├── tasks
│   └── instruntask.yml
└── vars
    ├── apachevars.yml
    └── mariadbvars.yml
```

```
[ansible@master01 base]$ vim vars/apachevars.yml
package: httpd
service: httpd
state: started
```

```
[ansible@master01 base]$ vim vars/mariadbvars.yml
package: "mariadb-server,MySQL-python"
service: "mariadb"
state: started
mysql_port: "3306"
dbuser: "elies"
dbname: "mabase"
upassword: "password"
```

```
[ansible@master01 base]$ vim tasks/instruntask.yml
# tasks/instruntask.yml
- name: Installs the {{ package }} package
  yum:
    name: "{{ package }}"
    state: latest
- name: Starts the {{ service }} service
  service:
    name: "{{ service }}"
    state: "{{ state }}"
```

```
[ansible@master01 base]$ vim multiplay2.yml
---
# handlers includes debug
################################################
##               Apache                       #
################################################
- name: Install start and enaible Apache
  hosts: host01
  tasks:
  - name: Includes service package and state Varaibles for Apache
    include_vars: vars/apachevars.yml

  - debug:
      msg: Chargement des variables pour Apache

  - name: Include install and start tasks
```

```
    include: tasks/instruntask.yml
    register: output
# add lineinfile to index.html then call handler restart
  - name: Add index.html to DocumentRoot
    lineinfile:
      dest: /var/www/html/index.html
      line: "{{ service }} is running"
      create: yes
    register: output
    notify: restart {{ service }}
  - debug:
      var: output
  - debug:
      var: output

  - name: Ensure the webserver is reachable
    uri:
      url: http://{{ ansible_hostname }}
      status_code: 200
    register: link
  - debug:
      var: link
################################################
#               Mariadb                        #
################################################
- name: Install start and enable Mariadb
  hosts: host01
#  debugger: on_skipped
  tasks:
  - name: Includes service package and state Varaibles for Mariadb
    include_vars: vars/mariadbvars.yml
  - name: Include install and start tasks
    include: tasks/instruntask.yml
# Configurer la base
  - name: creer la base de donnee
    mysql_db:
      name: "{{ dbname }}"
      state: present
  - name: creer un utilisateur
    mysql_user:
      name: "{{ dbuser }}"
      password: "{{ upassword }}"
      priv: "*.*:ALL "
      host: "localhost"
      state: present

# add lineinfile to index.html then call handler restart
  - name: Add bind-address to [mysqld] section in my.cnf file
    lineinfile:
      dest: /etc/my.cnf
      insertafter: '^\[mysqld\]'
      line: bind-address = 127.0.0.1
    notify: restart {{ service }}

  handlers:
    - name: restart {{ service }}
      service:
        name: "{{ service }}"
        state: restarted
```

```
[ansible@master01 base]$ ansible-playbook --syntax-check multiplay2.yml

[ansible@master01 base]$ ansible-playbook  multiplay2.yml
```