Lab 4 – Ansible Playbooks

Introduction

Les playbooks sont l'une des fonctionnalités principales de Ansible et indiquent à Ansible quoi exécuter. Ils sont comme une liste de tâches; chaque tâche est liée en interne à un bout de code appelé module.

Les playbooks sont des fichiers YAML simples et lisibles par les humains, tandis que les modules peuvent être écrits dans n'importe quel langage à condition que sa sortie soit au format JSON.

Vous pouvez avoir plusieurs tâches répertoriées dans un playbook et ces tâches seront exécutées en série par Ansible. Vous pouvez considérer les playbooks comme un équivalent des manifestes dans Puppet, des états dans Salt ou des recipies dans Chef; ils vous permettent de saisir une liste de tâches ou de commandes que vous souhaitez exécuter sur votre système distant.

Structure du playbook

Les Playbooks sont des fichiers texte écrits au format YAML et ont donc besoin de:

- commencer avec trois tirets (---)
- Indentation correcte en utilisant des espaces et **non des tabulations**!

Il y a quelques concepts importants:

- **hosts**: les hôtes gérés pour effectuer les tâches
- **tasks**: les opérations à effectuer en appelant les modules Ansible et en leur transmettant les options nécessaires.
- **become**: l'escalade de privilèges dans les Playbooks, identique à l'utilisation de -b dans la commande ad-hoc.

L'ordre des contenus dans un Playbook est important, car Ansible exécute les tâches dans l'ordre dans lequel ils sont présentés.

Un Playbook doit être **idempotent**. Par conséquent, si un Playbook est exécuté une fois pour mettre les hôtes dans le bon état, on doit être sûr que l'exécuter une seconde fois ne devrait plus apporter de modifications aux hôtes.

La plupart des modules Ansible sont idempotents, il est donc relativement facile de s'assurer que cela est vrai.

Essayez d'éviter les modules `shell` et `raw` dans les Playbooks. Comme ils prennent des commandes arbitraires, il est très facile de se retrouver avec des Playbooks non idempotents avec ces modules.

Votre premier Playbook

Il est temps de créer votre premier Playbook. Dans cet atelier, vous créez un playbook pour configurer un serveur Web Apache en trois étapes:

1. Première étape: installer le paquet **httpd**
2. Deuxième étape: Activer / démarrer le **service httpd**
3. Troisième étape: créer un fichier **index.html**

Playbook: Install Apache

Ce Playbook vérifie que le paquet contenant le serveur Web Apache est installé sur **les serveurs web**. Vous devez évidemment utiliser l'escalade de privilèges pour installer un package ou exécuter toute autre tâche nécessitant des autorisations `root`. Cela se fait dans le Playbook avec **`become: yes`**.

Sur **master** en tant qu'utilisateur `vagrant`, créez le fichier `apache.yml` avec le contenu suivant:

```
---

- name: Apache server installed
  hosts: webservers
  become: yes
  tasks:
  - name: latest Apache version installed
    yum:
      name: httpd
      state: latest
```

Cela montre l'un des points forts de Ansible: la syntaxe du Playbook est facile à lire et à comprendre. Dans ce playbook:

- Un nom est donné à la **play**
- Les hôtes cibles et l'escalade des privilèges sont configurées
- Une tâche est définie et nommée, elle utilise ici le module `yum` avec les options nécessaires.

Exécution du playbook

Les playbooks sont exécutés à l'aide de la commande `ansible-playbook` sur le nœud de contrôle **master**. Avant de lancer un nouveau Playbook, il est conseillé de vérifier les erreurs de syntaxe:

```
$ ansible-playbook --syntax-check apache.yml
```

Vous devriez maintenant être prêt à exécuter votre Playbook:

```
$ ansible-playbook apache.yml
```

Exécutez le Playbook une seconde fois. Les différentes couleurs, les compteurs **"ok"**, **"changed"** et **"PLAY RECAP"** permettent de repérer facilement ce que Ansible a réellement fait.

Étendre votre Playbook: Démarrer et activer Apache

La partie suivante du Playbook vérifie que le serveur Web Apache est *activé* et *démarré* sur **les serveurs web**.

Sur **master** , éditez le fichier `apache.yml` pour ajouter une seconde tâche à l'aide du module **service**. Le Playbook doit maintenant ressembler à ceci:

```
---

- name: Apache server installed
  hosts: webservers
  become: yes
  tasks:
  - name: latest Apache version installed
    yum:
      name: httpd
      state: latest
  - name: Apache enabled and running
    systemd:
      name: httpd
      enabled: true
      state: started
```

Et encore une fois, il est facile à comprendre:

- une deuxième tâche est définie
- un module est spécifié (**systemd**)
- les options sont fournies

Exécutez votre Playbook étendu:

```
$ ansible-playbook apache.yml
```

Remarque: certaines tâches sont indiquées en vert par "ok" et une par "changed" en jaune. Utilisez à nouveau une commande ad-hoc pour vous assurer qu'Apache a été *activé* et *démarré*, par exemple: `systemctl status httpd`

Étendez votre Playbook: Créez un index.html

Alors pourquoi ne pas utiliser Ansible pour déployer un simple fichier *index.html*? Créez le fichier `index.html` sur le noeud de contrôle **master**:

```
<body>
<h1>Apache is running fine</h1>
</body>
```

Vous avez déjà utilisé le module `copy` de Ansible pour écrire le texte fourni sur la ligne de commande dans un fichier. Maintenant, vous utiliserez ce module dans votre Playbook pour copier un fichier: Sur

master en tant qu'utilisateur **vagrant**, éditez le fichier *apache.yml* et ajoutez une nouvelle tâche à l'aide du module *copy*. Cela devrait maintenant ressembler à ceci:

```
---

- name: Apache server installed
  hosts: webservers
  become: yes
  tasks:
  - name: latest Apache version installed
    yum:
      name: httpd
      state: latest

  - name: Apache enabled and running
    service:
      name: httpd
      enabled: true
      state: started

  - name: copy index.html
    copy:
      src: index.html
      dest: /var/www/html/
```

La nouvelle tâche utilise le module `copy` et définit les options de *source* et de *destination* pour l'opération de copie. Exécutez votre Playbook étendu:

```
$ ansible-playbook apache.yml
```

- Regardez bien la sortie
- Exécutez la commande ad hoc en utilisant le module "uri" pour tester à nouveau Apache. La commande doit maintenant renvoyer une ligne verte "status: 200", entre autres informations.

```
$ ansible -m uri -a "url=http://localhost/" app1
```

Exemple d'Output:

```
app1 | SUCCESS => {
"accept_ranges": "bytes",
"ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python"
},
"changed": false,
"connection": "close",
"content_length": "24",
"content_type": "text/html; charset=UTF-8",
"cookies": {},
"cookies_string": "",
"date": "Sat, 23 Jan 2021 06:32:09 GMT",
"elapsed": 0,
```

```
    "etag": "18-5b98b763ac54c",
    "last_modified": "Sat, 23 Jan 2021 06:31:58 GMT",
    "msg": "OK (24 bytes)",
    "redirected": false,
    "server": "Apache/2.4.6 (CentOS)",
    "status": 200,
    "url": "http://localhost/"
    }
```

Verbosité de Ansible

L'une des premières options que tout le monde choisit est l'option de débogage. Pour comprendre ce qui se passe lorsque vous exécutez le playbook, vous pouvez l'exécuter avec l'option verbose (**-v**). Chaque **v** supplémentaire fournira à l'utilisateur plus d'informations de débogage.

essayez avec:

- L'option **-v** fournit la sortie par défaut.
- L'option **-vv** ajoute un peu plus d'informations.
- L'option **-vvv** ajoute beaucoup plus d'informations.

Challenge

1. Ecrivez un playbook **unarchive_url.yml** qui permet de télécharger l'application **wordpress** et **l'extraire** dans le dossier **/var/www/html** du nœud **app1** avec les permissions 755

```
---
- name: Playbook to download and install tomcat8
  hosts: app2
  become: yes
  tasks:
  - name: latest Apache version installed
    yum:
      name: httpd
      state: latest

  - name: Apache enabled and running
    service:
      name: httpd
      enabled: true
      state: started

  - name: Unarchive Application Wordpress
    unarchive:
      src: https://wordpress.org/latest.tar.gz
      dest: /var/www/html/
      remote_src: yes
```

2. Ecrivez un playbook **mysql.yml** qui installe le serveur **mysql-server** sur le nœud **db**, **l'active** pour démarrer par défaut et crée un utilisateur **admin** et une base de données **sitedb**

Sur un serveur Centos

```
---
- name: Configuration du serveur MySQL
  hosts: db
  become: yes
  tasks:
  - name: Installer serveur MariaDB
    yum:
      name: mariadb-server
      state: present

  - name: Activation et demarrage du service mariadb-server
    systemd:
      name: mariadb
      state: started
      enabled: yes

  - name: installer MySQL-python
    yum:
      name: MySQL-python
      state: present

  - name: Create admin user'
    mysql_user:
      name: admin
      password: 12345
      priv: '*.*:ALL,GRANT'
      state: present

  - name: creation de la base des données
    mysql_db:
      login_user: admin
      login_password: 12345
      name: sitedb
      state: present
```

Sur un serveur Ubuntu

```
---
- name: install mysql
  hosts: ubuntu
  tasks:
  - name: Install python-mysqldb
    apt: python-mysqldb update_cache=yes cache_valid_time=3600 state=present
    become: yes

  - name: Install mysql-server
    apt: name=mysql-server update_cache=yes cache_valid_time=3600 state=present
    become: yes

  - name: Start the MySQL service
    become: yes
    service:
      name: mysql
      state: started
      enabled: true

  - name: update mysql root password for all root accounts
    become: yes
    mysql_user:
      name: root
      host: "localhost"
      password: "password"
      login_user: root
      login_password: "password"
      check_implicit_admin: yes
      priv: "*.*:ALL,GRANT"

  - name: create database
    become: yes
    mysql_db:
      name: mydb
      login_user: root
      login_password: "password"
```