EXÉCUTEZ VOS PREMIÈRES COMMANDES ANSIBLE

Introduction

Maintenant que nous avons étudié comment installer et configurer notre environnement Ansible, et que nous avons vu le fonctionnement global d'Ansible. Il est temps maintenant de **lancer nos premières commandes Ansible**.

Comme vu sur les chapitres précédents Ansible peut servir pour différentes tâches, et nous allons alors tout au long de ce cours **comprendre les cas d'utilisation les plus courants** avant d'explorer toutes les puissantes fonctionnalités de configuration et d'orchestration qu'offre Ansible.

Concernant mon environnement de travail pour ce cours, j'aurai comme IP statique `192.169.0.21` pour mon nœud distant numéro 1 et l'IP `192.168.0.22` pour mon nœud distant numéro 2. J'en ai aussi profité pour rajouter en outre un alias dans le fichier `/etc/hosts` afin de mieux distinguer mes différentes machines, voici à quoi ressemble le contenu de ce fichier :

```
192.168.0.21 slave-1
192.168.0.22 slave-2
```

Lancement de notre première commande Playbook

Prérequis

Ansible communique avec des machines distantes via le protocole SSH. Il faut donc vous assurez que vous pouvez vous connecter à vos différents nœuds distants de votre inventaire en utilisant le protocole SSH. Dans notre cas nous utiliserons la commande `ssh-copy-id` afin d'ajouter la clé SSH publique du serveur de contrôle au fichier **`authorized_keys`** sur nos systèmes gérés.

Par défaut, Vagrant vous donne l'accès en ssh depuis l'utilisateur `vagrant` avec le mot de passe `vagrant`. Débutons alors par la **configuration des accès de notre machine de contrôle** :

```
ssh-copy-id vagrant@192.168.0.21
# mdp : vagrant
ssh-copy-id vagrant@192.168.0.22
# mdp : vagrant
```

L'inventaire

Ansible lit les informations sur les machines que vous souhaitez gérer à partir d'un fichier nommé l'**inventaire**. Bien que vous puissiez transmettre une adresse IP en paramètre avec la commande `ansible`, vous aurez besoin d'un inventaire pour profiter de la flexibilité et de la répétabilité complètes d'Ansible.

Vous pouvez définir vos machines esclaves dans le fichier inventaire qui est situé par défaut dans le fichier **`/etc/ansible/host`**. Votre inventaire peut stocker les adresses IP ou les noms de domaines complets de vos machines distantes (mais aussi des groupes, des alias ou des variables).

Dans cet exemple, dans mon fichier inventaire je vais insérer le nom DNS complet ma machine esclave numéro 1 et l'IP de ma machine esclave numéro 2 :

```
slave-1
192.168.0.22
```

Nous verrons plus tard dans ce chapitre **comment intégrer la notion de groupes dans notre inventaire**.

Exécutez vos premières commandes Ansible

D'abord qu'est-ce qu'un module ? Les modules sont comme de petits programmes qu'Ansible pousse depuis une machine de contrôle vers tous les hôtes distants. Les modules sont exécutés à l'aide de playbooks ou depuis la cli Ansible, et ils contrôlent des éléments tels que les services, les packages, les fichiers et bien plus.

Dans notre exemple, on utilisera le module ping qui envoie une requête ping à tous les nœuds de votre inventaire. Ouvrez donc un terminal depuis votre machine de contrôle et lancez-y la commande suivante :

```
ansible all -m ping -u vagrant
```

Voici une liste d'explication des différentes options de notre commande :

- `all` : ici on demande à Ansible d'exécuter la commande sur tous les hôtes de notre inventaire
- `-m ping` : ici on lui demande d'utiliser le module ping
- `-u vagrant` : ici on lui demande de lancer notre module depuis l'utilisateur vagrant

Résultat :

```
192.168.0.22 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

```
slave-1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

Vous pouvez aussi **exécuter directement des commandes Linux sur Ansible**, en utilisant tout simplement l'option `-a`. Par exemple :

```
ansible all -a "whereis python"
```

Résultat :

```
192.168.0.22 | CHANGED | rc=0 >>
python: /usr/bin/python3.6 /usr/bin/python3.6m /usr/bin/python /usr/bin/python2.7 ...
slave-1 | CHANGED | rc=0 >>
python: /usr/bin/python3.6 /usr/bin/python3.6m /usr/bin/python /usr/bin/python2.7 ...
```

Certaines commandes ou modules peuvent nécessiter une **élévation de privilèges**. Par exemple si vous tentez d'exécuter la commande ci-dessous depuis le compte `vagrant` vous obtiendrez une jolie erreur Permission denied :

```
ansible all -a "grep vagrant /etc/shadow" -u vagrant
```

Erreur :

```
slave-1 | FAILED | rc=2 >>
grep: /etc/shadow: Permission denied non-zero return code

192.168.0.22 | FAILED | rc=2 >>
grep: /etc/shadow: Permission denied non-zero return code
```

Dans le cas où vous auriez besoin d'une élévation de privilèges pour l'exécution d'une commande, utilisez alors l'option `--become` (cette option exploite les outils existants d'élévation de privilèges comme `sudo`), comme suit :

```
ansible all -a "grep vagrant /etc/shadow" -u vagrant --become
```

Résultat :

```
slave-1 | CHANGED | rc=0 >>
vagrant:$6$lrFYJT5b$ZmlDfFD2P8FWgB....

192.168.0.22 | CHANGED | rc=0 >>
vagrant:$6$lrFYJT5b$ZmlDfFD2P8FWgB...
```

Les groupes dans l'inventaire

Vous pouvez (et le ferez probablement) mettre chaque hôte distant dans un ou plusieurs groupes. Par exemple, vous pouvez intégrer vos machines de production dans un groupe nommé `[prod]` et vos machines de test dans un groupe nommé `[dev]`.

Mettons en pratique notre exemple. Voici donc à quoi ressemblera notre fichier inventaire :

```
[prod]
slave-1
[dev]
192.168.0.22
192.168.0.23
```

Pour lancer nos modules ou commandes sur un groupe d'hôtes, il suffit de remplacer l'option `all` par le nom de notre groupe :

```
ansible dev -m ping -u vagrant
```

Résultat :

```
192.168.0.22 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
```

```
    "ping": "pong"
}

192.168.0.23 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

Voici la commande pour **lancer vos modules sur différents groupes de votre inventaire Ansible**. Exemple sur le groupe `dev` et `prod` :

```
ansible dev,prod -m ping -u vagrant
```

Utilisateur par défaut

Je ne sais pas vous, mais de mon côté j'en ai marre de taper à chaque fois l'option `-u vagrant`, j'aimerais bien que cette option soit prise en charge par défaut lors de l'exécution des modules/commandes Ansible. Ça tombe bien, car il est possible de définir un utilisateur par défaut sur votre inventaire avec l'option `ansible_user` :

```
slave-1 ansible_user=vagrant
192.168.0.22 ansible_user=vagrant
```

Dorénavant, il n'y a plus besoin d'utiliser l'option `-u` :

```
ansible all -m ping
```

Résultat :

```
192.168.0.21 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

```
192.168.0.22 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
```

Les modules avec les options

Certains modules ansible nécessitent des paramètres supplémentaires pour une utilisation efficiente du module en question. Par exemple si on fait un tour sur la documentation du module file (module permettant de gérer les fichiers et les propriétés des fichiers.) on peut remarquer qu'il peut être utilisé avec différents paramètres additionnels :

Dans notre exemple on souhaite créer un fichier vide nommé **test.txt** dans le dossier **/home/vagrant/** et ne lui fournir que des droits de lecture (0644).

- **`path`** (obligatoire) : pour l'emplacement et le nom de notre fichier (chemin absolu + le nom de notre fichier)
- **`state`** : nous utiliserons la valeur **`touch`** pour obliger la création de notre fichier. message personnalisé et le paramètre
- **`mode`** : pour les autorisations que le fichier doit avoir dans notre cas ça sera le droit 0644

Pour cela, nous utiliserons l'option **`-a`** qui permet également de fournir des arguments complémentaires à l'option **`-m`** (module). Nous aurons ainsi la commande suivante :

```
ansible all -m file -a "path=/home/vagrant/test.txt state=touch mode=0644"
```

Résultat :

```
192.168.0.21 | CHANGED => {
    "changed": true,
    "dest": "/home/vagrant/test.txt",
    "gid": 0,
    "group": "vagrant",
    "mode": "0644",
    "owner": "vagrant",
    "secontext": "unconfined_u:object_r:user_home_t:s0",
    "size": 0,
    "state": "file",
    "uid": 0
}

192.168.0.22 | CHANGED => {
    "changed": true,
    "dest": "/home/vagrant/test.txt",
    "gid": 0,
    "group": "vagrant",
    "mode": "0644",
    "owner": "vagrant",
    "secontext": "unconfined_u:object_r:user_home_t:s0",
    "size": 0,
    "state": "file",
    "uid": 0
}
```

Parallélisme

Par défaut, Ansible utilise seulement 5 processus simultanés. Cependant, si vous avez plus de hôtes que la valeur définie par défaut pour le nombre de processus, Ansible prendra ainsi un peu plus de temps pour traiter tous vos hôtes. Vous pouvez augmenter le nombre de processus avec l'option `-f`. Dans cet exemple je vais utiliser 9 processus simultanés pour lancer le module ping :

```
ansible all -m ping -f 8
```

Recueillir des informations (Facts) sur vos hôtes

Dans cette partie, nous allons **passer en revue les Facts sur Ansible** et comment vous pouvez les utiliser avec la commande `ansible`.

Les Facts sont des propriétés système qui sont collectées par Ansible lors de son exécution sur un système distant. Les Facts contiennent des détails utiles tels que le stockage, la version (mineur et majeur) de l'OS, configuration du réseau sur un système cible. Ils peuvent être exportés vers un fichier en tant que type de rapport système, ou ils peuvent être utilisés pendant l'exécution d'un playbook Ansible pour conditionner l'exécution de vos tâches.

Voici la commande pour voir tous les Facts :

```
ansible all -m setup
```

Vous pouvez également filtrer la sortie pour afficher uniquement certains Facts en utilisant le paramètre `filter`. Dans cet exemple je vais récupérer des informations réseau de mes hosts :

```
ansible all -m setup -a 'filter=*ip*'
```

Résultat :

```
192.168.0.21 | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "192.168.0.21"
        ],
        "ansible_all_ipv6_addresses": [],
        "ansible_default_ipv4": {
            "address": "192.168.0.21",
            ...
            "gateway": "192.168.0.1",
            "interface": "wlp1s0",
            "macaddress": "00:40:56:94:5f:8e",
            "netmask": "255.255.255.0",
            ...
        },
        ...
```

Pour plus d'informations, consultez la documentation du module setup.

Conclusion

Nous nous sommes penchés dans ce chapitre sur l'exécution des modules et des commandes depuis la cli d'ansible. Vous êtes désormais capable de vous amuser avec les différents modules qu'offre Ansible. Dans notre futur chapitre nous apprendrons à créer et à exécuter notre premier playbook Ansible, ce qui nous facilitera ainsi la gestion de nos modules.