Lab 4 – Variables

Introduction

Ansible prend en charge les variables permettant de stocker des valeurs pouvant être utilisées dans les Playbooks. Les variables peuvent être définies à divers endroits et ont une priorité claire. Ansible remplace la variable par sa valeur lorsqu'une tâche est exécutée.

Les variables sont référencées dans les Playbooks en plaçant le nom de la variable entre deux accolades. Voici une variable `{{ variable1 }}`

La pratique recommandée est de définir des variables dans des fichiers situés dans deux répertoires nommés `host_vars` et `group_vars`:

Par exemple pour définir des variables pour le groupe "servers", créez un fichier YAML nommé `group_vars/servers` avec les définitions de variables.

Pour définir des variables spécifiquement pour l'hôte "host1.example.com", créez le fichier `host_vars/host1.example.com` avec les définitions de variables.

Les variables hôtes ont priorité sur les variables de groupe (pour plus d'informations sur la priorité, reportez-vous à la documentation).

Managing Variables

Nous pouvons définir des variables à différents endroits dans un projet Ansible.

- **Global Scope** - Variables définies à partir de l'interface de ligne de commande ou de la configuration Ansible (**ansible.cfg**).
- **Play Scope** - Les variables sont définies dans le playbook.
- **Host Scope** - Variables définies sur des groupes d'hôtes ou des hôtes individuelles par l'inventaire, fact gathering, etc.

Définir des variables dans les fichiers de configuration (ansible.cfg)

Nous commençons par définir des variables globales dans un fichier *ansible.cfg* dans le répertoire de travail local.

```
[master]$ vi ansible.cfg
```

```
[defaults]
inventory = hosts
host_key_checking = False
```

Ces variables seront prises en considération à chaque exécution d'un playbook à partir de ce répertoire.

créons notre fichier *hosts* contenant la liste des noeuds:

```
[master]$ vi hosts
```

```
[webserver]
centos01 ansible_host=10.0.0.21
centos02 ansible_host=10.0.0.22
centos03 ansible_host=10.0.0.23

[prod]
centos01 ansible_host=10.0.0.21
```

Définir des variables dans les playbooks

Les administrateurs peuvent définir leurs propres variables dans les Playbooks et les utiliser dans des tâches.

Les variables de playbooks peuvent être définies de plusieurs manières:

- En les plaçant directement dans un bloc vars au début d'un Playbook:

```
- hosts: all
  vars:
    user: joe
    home: /home/joe
```

- En définissant les variables de Playbook dans des fichiers externes:

```
- hosts: all
  vars_files:
  - vars/users.yml
```

- Les variables sont ensuite définies dans ce fichier au format YAML:

```
---
user: joe
home: /home/joe
```

Pour mieux comprendre, faisons une application. Modifiez le fichier index.html pour afficher l'environnement de développement (**dev / prod**) dans lequel un serveur est déployé.

Sur **le serveur de contrôle** en tant qu'utilisateur ***devops***, créez les répertoires contenant les définitions de variable **`host`** et **`group`**:

```
$ mkdir {host_vars,group_vars}
```

Créer les fichiers variables

Créez maintenant deux fichiers contenant des définitions de variables qui pointent vers un environnement:

- ~/ansible/group_vars/webserver contenant:

```
---
stage: dev
```

- ~/ansible/host_vars/centos01 , contenant:

```
---
stage: prod
```

Créer des fichiers index.html

Créez deux fichiers sous ~/ansible/ :

1. Un appelé ***prod_index.html*** avec le contenu suivant:

```
<body>
<h1>This is a production webserver, take care!</h1>
</body>
```

2. Et l'autre appelé ***dev_index.html*** avec le contenu suivant:

```
<body>
<h1>This is a development webserver, have fun!</h1>
</body>
```

Créer le playbook

Vous avez maintenant besoin d'un Playbook qui copie le fichier prod ou dev_index.html en fonction de la variable "**stage**".

Créez un nouveau Playbook appelé `deploy_index_html.yml` dans le répertoire `~/ansible/`.

Notez comment la variable "**stage**" est utilisée dans le nom du fichier à copier.

```
---
- name: Copy index.html
  hosts: webserver
  become: yes
  tasks:
  - name: copy index.html
    copy:
      src: "{{ stage }}_index.html"
      dest: /var/www/html/index.html
```

Exécutez le Playbook:

```
$ ansible-playbook deploy_index_html.yml
```

Tester le résultat

Le Playbook doit copier différents fichiers sous le nom index.html sur les hôtes, utilisez curl pour le tester:

```
[master]$ curl 10.0.0.21
<body>
<h1>This is a development webserver, have fun!</h1>
</body>
```

```
[master]$ curl 10.0.0.22
<body>
<h1>This is a production webserver, take care!</h1>
</body>
```

Peut-être vous pensez maintenant: il doit y avoir un moyen plus intelligent de modifier le contenu des fichiers... vous avez absolument raison. Ce lab a été conçu pour introduire les variables. Vous êtes sur le point de connaître les modèles (templates) dans l'un des prochains labs.

Challenge Lab: Utilisation des variables d'inventaire

Vous avez maintenant toutes les informations pour effectuer les tâches suivantes:

1. Dans le fichier de variables du groupe **webserver**, définissez une variable **service** par *sshd*.
2. Dans le fichier de variable hôte de l'hôte **centos01**, définissez une variable **service** par *httpd*.
3. Créez un playbook `check_service.yml` pour redémarrer un service dont le nom est défini par la variable **service**. Rendez-le applicable à toutes hôtes **centos**.
4. Exécutez le Playbook avec "-v" pour voir Ansible vérifie différents services en fonction des variables.

Solution ci-dessous

```
[master]$ cat host_vars/centos01
---
stage: prod
service: httpd
```

```
[master]$ cat group_vars/webserver
---
stage: dev
service: sshd

[master]$ cat check_service.yml
```

```
---
- name: Check if service is enabled and started
  hosts: centos0*
  become: yes
  tasks:
  - name: Check service is enabled and started
    service:
    name: "{{ service }}"
    enabled: true
    state: started
```

```
[master]$ ansible-playbook check_service.yml -v
```

Variables enregistrées

Les administrateurs peuvent capturer le résultat d'une commande à l'aide de l'instruction `register`. La sortie est enregistrée dans une variable qui pourra être utilisée ultérieurement, soit à des fins de débogage, soit pour réaliser autre chose, telle qu'une configuration particulière basée sur la sortie d'une commande.

Le Playbook suivant montre cette utilisation, créez-le en tant que `register_var.yml` et exécutez-le:

```
---
- name: Installs a package and prints the result
  hosts: ubuntu
  become: True
  tasks:
  - name: Install the package
    apt:
      name: apache2
      state: present
    register: install_result

  - debug: var=install_result
```

Quand ce Playbook est lancé:

- le module de débogage est utilisé pour vider la valeur de la variable enregistrée *install_result* dans le terminal.
- Cela peut être très utile lors du débogage, nous en parlerons plus tard. Dans ce cas, le paquet était déjà installé et est reflété par la sortie du module capturée dans la variable.

Ansible Facts

Ansible Facts sont des variables automatiquement découvertes par Ansible à partir d'une hôte gérée. **Facts** sont extraits par le module `setup` et contiennent des informations utiles stockées dans des variables que les administrateurs peuvent réutiliser.

Pour avoir une idée des **facts** recueillis par Ansible par défaut, sur **master** en tant qu'utilisateur vagrant, exécutez:

```
[master]$ ansible centos01 -m setup
```

C'est peut-être un peu trop, vous pouvez utiliser des *filtres* pour limiter la sortie à certains faits, l'expression est un joker de style shell:

```
[master]$ ansible centos01 -m setup -a 'filter=ansible_eth0'
```

Ou que diriez-vous de rechercher uniquement des faits liés à la mémoire:

```
[master]$ ansible all -m setup -a 'filter=ansible_*_mb'
```

Challenge Lab: Facts

Essayez de trouver et d'imprimer la distribution (CentOS) de vos hôtes gérées sur une seule ligne. Utilisez `grep` pour trouver le **Fact**, puis appliquez un filtre pour n'imprimer que ce fact.

Solution

```
[master]$ ansible centos01 -m setup | grep distribution
```

```
[master]$ ansible all -m setup -a 'filter=ansible_distribution' -o
```

Utilisation des facts dans les playbooks

Les **facts** peuvent être utilisés dans un Playbook comme des variables, en utilisant le nom approprié. Créez le Playbook *facts.yml*:

```
---
- name: Output facts within a playbook
  hosts: centos
  tasks:
  - name: Prints Ansible facts
    debug:
      msg: The default IPv4 address of {{ ansible_fqdn }} is {{
ansible_eth1.ipv4.address }}
```

Exécutez-le pour voir comment les **facts** sont imprimés:

```
[master]$ ansible-playbook facts.yml
PLAY [Output facts within a playbook]
*********************************
TASK [Gathering Facts]
*********************************************
ok: [centos01]
ok: [centos02]
ok: [centos03]
TASK [Prints Ansible facts]
*****************************************
ok: [centos01] => {
"msg": "The default IPv4 address of slave01 is 10.0.0.21"
}

ok: [centos02] => {
"msg": "The default IPv4 address of slave02 is 10.0.0.22"
}
ok: [centos03] => {
"msg": "The default IPv4 address of slave03 is 10.0.0.23"
}
PLAY RECAP
************************************************************

centos01 : ok=2 changed=0 unreachable=0 failed=0 skipped=0 rescued=0
ignored=0
centos02 : ok=2 changed=0 unreachable=0 failed=0 skipped=0 rescued=0
ignored=0
centos03 : ok=2 changed=0 unreachable=0 failed=0 skipped=0 rescued=0
ignored=0
```

Notez comment le module **setup** a été appelé implicitement par Ansible sous le nom de tâche "Gathering Facts".