Installer, tester et déployer une solution d'infrastructure réseau

Découverte d'Ansible

Installation d’Ansible sur Ubuntu Server 20.04

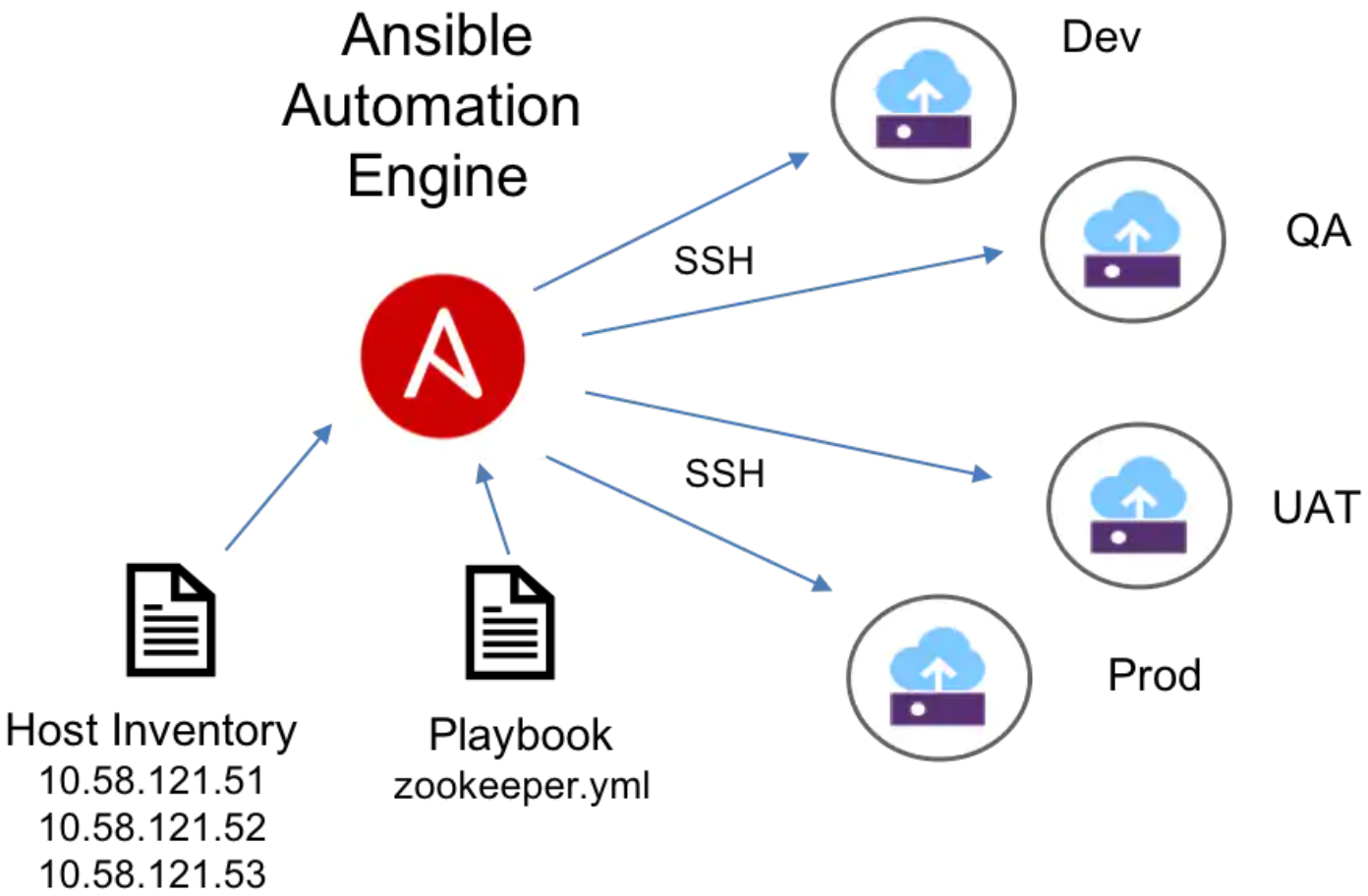
1. Présentation de l’outil

1.1. Présentation générale

Ansible est un moteur d'automatisation simple qui automatise le cloud, le provisionnement, la gestion de la configuration, le déploiement d'applications, l’orchestration de services ainsi que beaucoup d’autres fonctionnalités. C’est un logiciel open source développé en langage Python. Les modules Ansible sont regroupés en fonction des opérations qu'ils effectuent: des modules de Cloud, des modules de commandes, des modules de base de données, des modules de réseau, etc.

1.2. Architecture

- Le contrôleur Ansible possède des modules pour effectuer différentes opérations sur le(s) serveur cible.
- Aucun agent ou logiciel requis sur le serveur cible (serveur managé).
- Une connectivité via SSH aux machines cibles est utilisée pour effectuer les opérations et recueillir les résultats de retour.
- Ansible utilise un langage très simple (YAML dans les Playbooks) qui est presque comme un anglais simple.



1.3. L'inventaire

L'Inventaire fournit la liste des hôtes (serveurs managés) sur lesquels Ansible exécutera les tâches. Il peut également être utilisé pour regrouper les hôtes et configurer les variables pour les hôtes et les groupes comme ci-dessous.

L'inventaire peut être configuré comme un fichier plat (host/etc/ansible/) ou configuré à partir de sources dynamiques telles que VMware vCenter ou les fournisseurs de cloud.

1.4. Les Commandes Ad-Hoc

Les commandes Ad-Hoc sont utilisées pour vérifier rapidement le fonctionnement et pas besoin d'être enregistrés (comme les Playbooks) pour une utilisation ultérieure. Par exemple, si un redémarrage soudain est nécessaire sur des centaines de serveurs ou s'il est nécessaire de vérifier la disponibilité des serveurs sur le réseau, les commandes ad hoc peuvent être utilisées.

Une documentation complète des commandes Ad-Hoc ainsi que pour leurs utilisation est disponible sur le site Ansible: http://docs.ansible.com/ansible/intro_adhoc.html

1.5. Le Playbook Ansible

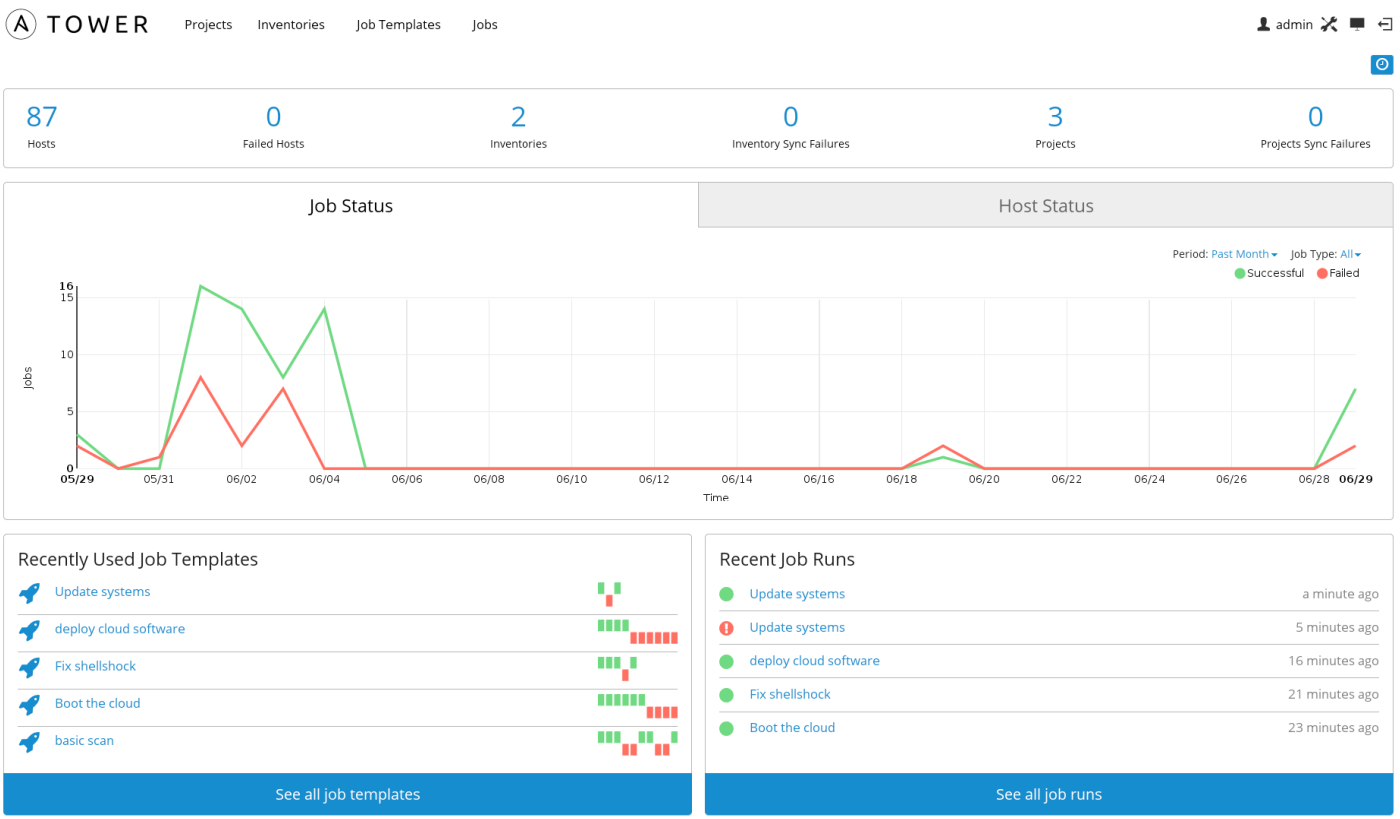
Le Playbook est le langage de configuration d'Ansible utilisé pour effectuer l'ensemble des tâches et d'étapes de configuration ou pour faire appliquer une politique sur les nœuds distants. Les modules ansible sont utilisés dans le Playbook pour exécuter l'opération.

Le Playbook est écrit en YAML qui est un langage simple, lisible par l'homme et développé dans un anglais de base comme la langue du texte. Les Playbooks sont plus susceptibles d'être conservés dans le système de contrôle de version et utilisés pour assurer les configurations des systèmes distants.

Les Playbooks sont utilisés de la simple configuration et la gestion des nœuds distants jusqu'au déploiement avancé impliquant des mises à jour, le suivi des serveurs, la répartition de charge, etc. Une tâche Ansible peut être simplement représentée comme un appel au module Ansible. Une lecture Ansible (Play) contient un groupe d'hôtes et un ensemble de tâches. Un Playbook peut contenir une ou plusieurs Plays.

1.6. Ansible Tower

Ansible Tower est une interface Web utilisateur facile à utiliser pour Ansible. Il contient les fonctions ansible les plus importantes. L'application centralise l'infrastructure avec contrôle d'accès basé sur les rôles, la planification des tâches et la gestion des stocks graphiques. L'API et le CLI REST de Ansible Tower font qu'il est facile à intégrer dans les outils existants. Cependant, Tower est un produit payant, qui nécessite une licence. Une version d'essai gratuite est mise à disposition pour tester le produit.



2. Débuter avec Ansible

Ansible utilise une simple connexion SSH pour réaliser les différentes actions à mener. Ces actions sont codées dans des playbooks grâce à la syntaxe YAML. l'avantage majeur de ce langage est qu'il est extrêmement simple à prendre en main.

Voici quelques terminologies qui peuvent vous être utiles :

- Control machine : la machine maître, celle depuis laquelle nous lancerons nos playbooks;
- Host Inventory : les hôtes sur lesquels seront lancées les actions;
- Playbooks : fichier décrivant des tâches qui seront lancées sur nos hôtes;
- Tâches: ce sont les actions que nous ferons sur nos hôtes (ex: installer un package)
- Rôles: ensemble de tâches réparties dans différents dossiers. Ces dossiers seront des rôles;
- Modules: les modules seront appelés par des tâches. Ce sont les modules qui permettent d'exécuter une action (pt, copy, cmd...).

Vous trouverez la liste des de tous les modules existants via le lien suivant : https://docs.ansible.com/ansible/latest/collections/index.html#list-of-collections

3. Installation de l'application Ansible

Commençons par installer un serveur Ansible via les opérations suivantes :

3.1. Mise à jour du serveur

Dans un premier temps, mettons à jour notre serveur :

```
sudo apt-get update
sudo apt-get upgrade -y
```

Puis redémarrez votre VM si nécessaire.

3.2. Installation des pré-requis technique et d'Ansible

Puis installons Ansible depuis le dépôt :

```
sudo apt install software-properties-common
sudo apt-add-repository --yes --update ppa:ansible/ansible
sudo apt install ansible -y
```

3.3. Vérification de la version

Lorsque l'installation est réalisée, vous pouvez vous assurer que celui-ci est bien installé via la commande :

```
ansible --version
```

Vous devez obtenir un résultat similaire à celui-ci :

```
roger@ubuntu-preprod:~$ ansible --version
ansible 2.9.5
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/home/roger/.ansible/plugins/modules',
u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/local/lib/python2.7/dist-packages/ansible
  executable location = /usr/local/bin/ansible
  python version = 2.7.17 (default, Nov  7 2019, 10:07:09) [GCC 7.4.0]
```

Vous trouverez la configuration d'Ansible dans le répertoire */etc/ansible*.

Par défaut, voici les fichiers présents dans ce répertoire :

```
roger@ubuntu-preprod:~$ ls -l /etc/ansible/
total 28
-rw-r--r-- 1 root root 19985 janv. 20 21:11 ansible.cfg
-rw-r--r-- 1 root root  1016 janv. 20 21:11 hosts
drwxr-xr-x 2 root root  4096 janv. 20 21:12 roles
```

- hosts : Il s'agit du fichier qui permet de spécifier les machines clientes, celles qui pourront être assujetties aux commandes Ansible et à une gestion au travers Ansible. Également, ce fichier permet de spécifier des groupes de machines, afin d'accélérer la gestion de plusieurs machines disposant d'une même configuration par exemple
- ansible.cfg : Il s'agit de la configuration principale d'Ansible, commandes, modules, plugins et configuration SSH s'y trouvent
- roles : ce dossier nous servira surtout par la suite quand on commencera à avoir plusieurs rôles à nos playbooks.

4. Configuration de l'accès SSH au serveur

Ansible s'appuie principalement sur le protocole SSH pour réaliser les opérations qu'on lui demande. Il est donc important de bien configurer son accès SSH qui sera utilisé sur l'ensemble des serveurs que gère Ansible.

Normalement, vous avez déjà généré une clé SSH pour votre compte utilisateur sur votre VM. Pour vérifier si c'est le cas, essayez de l'afficher via la ligne de commande suivante :

```
cat ~/.ssh/id_rsa.pub
```

Vous devez obtenir un résultat similaire à celui-ci :

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQC8MCgdrueOhTHjZSREaSbmXOE88b4et78KQvsbu/16/jKaBjkBsiig501Mydpzsnf
sT/h1/rstwo+MvCLxN83x176bnMFErFf4/0Gokjwt41KoEfGQ7/GMFko4vKAp6yE+wjmbS6JSoMtu/7p4wUSzoK1FAgXMFH
Gdcmf8sfJbUQNsaOWK3oWEDKWzF3jAV4UWaHuHVDcUUUMkN+heSr1cDc0r88dqhyadMDVT1ns7rm/2wBJPAHG0QaAizeJMX
6cvIMWnHQlGU+DAX0WoNAw4sRlkk7RNvPNYcsM+juEFBMct0pknWWcK56d7WWniMB/twk5bPmmGhbXvPQuqBa35
roger@ubuntu-preprod
```

Sinon, vous devez générer votre clé SSH via la commande suivante :

```
ssh-keygen -t rsa
```

Ensuite il faut envoyer la clé publique à l'ensemble des serveurs qui seront gérés par Ansible. Pour ce TP, on peut travailler sur la même machine et/ou travailler sur une seconde VM. La syntaxe de la commande est la suivante :

```
ssh-copy-id username@adresse_ip_de_votre_serveur_à_configurer
```

/!\ Attention : la commande ci-dessus est à personnaliser avec votre nom d'utilisateur sur votre VM.

Si l'opération se déroule correctement, vous devez obtenir un résultat similaire à celui-ci :

```
roger@ubuntu-preprod:~$ ssh-copy-id roger@192.168.1.69
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/roger/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that
are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to
install the new keys
roger@192.168.1.69's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'roger@192.168.1.69'"
and check to make sure that only the key(s) you wanted were added.
```

Pour ceux qui disposent d'un second serveur fonctionnel **UbuntuProd** par exemple, vous pouvez ajouter cette ligne de commande afin de déclarer votre clé SSH sur le serveur.

ssh-copy-id username@adresse_ip_du_serveur

5. Configuration du fichier hosts d'Ansible

Afin de pouvoir gérer nos serveurs, nous allons les ajouter dans la liste des hôtes pris en charge par Ansible. Il s'agit du fichier */etc/ansible/hosts*. Nous allons créer un groupe de machine, dénommé "webservers". Ce groupe sera composé des machines sur lesquelles vous avez déployé votre clé SSH.

Modifiez les lignes correspondantes afin de renseigner les adresses IP des serveurs pris en charge par Ansible.

```
# Ex 2: A collection of hosts belonging to the 'webservers' group
[webservers]
## alpha.example.org
## beta.example.org
127.0.0.1       #Adresse locale.
192.168.1.69    #Adresse ip du serveur UbuntuProd pour ceux qui en ont un fonctionnel.
```

Attention, pour ceux qui n'utilise pas le port 22 par défaut pour l'accès SSH, il faut l'indiquer dans le fichier de configuration

127.0.0.1 ansible_port=8022 #Exemple de connexion sur le port 8022.

Ainsi, lorsqu'avec Ansible on décidera de lancer une action sur le groupe "*webservers*", tous les serveurs composant ce groupe seront ciblés.

Vous pouvez tester votre configuration en faisant un ping sur les serveurs déclarés précédemment :

```
ansible -m ping all
```

Vous devez obtenir un résultat similaire à celui-ci :

```
roger@ubuntu-preprod:/etc/ansible$ ansible -m ping all
127.0.0.1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
192.168.1.69 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

Dans la commande que vous venez d'exécuter, l'argument ***all*** signifie tous les hôtes répertoriés. Mais il est également possible de :

- spécifier un groupe de serveurs : ***ansible server_group -m ping***
- spécifier un serveur uniquement : ***ansible -m ping server_name***
- spécifier plusieurs serveurs en les séparant par des deux points : ***ansible -m ping server_name1:server_name2***

Il est intéressant de savoir qu'Ansible permet d'utiliser des modules facilitant l'utilisation des fonctionnalités de votre distribution Linux. Par exemple :

- Le module "shell" permet de passer des commandes bash au travers Ansible, par exemple : ansible webservers -m shell -a "ls -l /home/"
- Le module "ping" permet de tester l'envoi de commande via Ansible en effectuant un ping, par exemple : ansible webservers -m ping
- Le module "apt" permet la gestion des paquets comme la commande "apt-get", par exemple : ansible webservers -m apt -a name=tree

Vous trouverez la liste des modules existants sur le site officiel d'Ansible : Modules Ansible.

5.1. Ansible CLI : la ligne de commandes

Tout d'abord, sachez que vous pouvez utiliser Ansible de deux manières :

- en ligne de commandes (avec la commande ansible)
- à partir des playbooks (avec la commande ansible-playbook)

A la différence des Playbooks, la ligne de commandes Ansible est utilisée pour définir et exécuter une seule tâche sur un ou plusieurs hôtes. Nous ferons ainsi du one-shot. Par exemple copier un fichier sur des serveurs, rebooter un groupe de serveurs, etc. Mais on ne l'utilisera pas qu'une seule fois, pour un besoin plutôt ponctuel.

Testons les commandes citées ci-dessus :

```
roger@ubuntu-preprod:/etc/ansible$ ansible webservers -m shell -a "ls -l /home/"
127.0.0.1 | CHANGED | rc=0 >>
total 8
drwxr-xr-x  6 deployer deployer 4096 févr. 16 12:41 deployer
drwxr-xr-x 22 roger    roger    4096 févr. 23 13:41 roger
192.168.1.69 | CHANGED | rc=0 >>
total 8
drwxr-xr-x 7 deployer deployer 4096 févr. 22 17:21 deployer
drwxr-xr-x 8 roger    roger    4096 févr. 23 13:34 roger
```

```
roger@ubuntu-preprod:/etc/ansible$ ansible webservers -m ping
127.0.0.1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
```

```
}
192.168.1.69 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

```
roger@ubuntu-preprod:/etc/ansible$ ansible webservers -m shell -a "uptime"
127.0.0.1 | CHANGED | rc=0 >>
 14:07:43 up  3:36,  2 users,  load average: 0,14, 0,05, 0,01
192.168.1.69 | CHANGED | rc=0 >>
 14:07:43 up  3:36,  2 users,  load average: 0,00, 0,00, 0,00
```

Vérification de la taille des filesystem du groupe de serveur “webservers” :

```
roger@ubuntu-preprod:/etc/ansible$ ansible webservers -m shell -a 'df -h'
192.168.1.69 | CHANGED | rc=0 >>
Filesystem      Size  Used Avail Use% Mounted on
udev            966M     0  966M   0% /dev
tmpfs           200M 1004K  199M   1% /run
/dev/sda2       9,8G  5,2G  4,1G  57% /
tmpfs           997M     0  997M   0% /dev/shm
tmpfs           5,0M     0  5,0M   0% /run/lock
tmpfs           997M     0  997M   0% /sys/fs/cgroup
/dev/loop1       92M   92M     0 100% /snap/core/8592
/dev/loop0       92M   92M     0 100% /snap/core/8689
tmpfs           200M     0  200M   0% /run/user/1000
127.0.0.1 | CHANGED | rc=0 >>
Filesystem      Size  Used Avail Use% Mounted on
udev            1,9G     0  1,9G   0% /dev
tmpfs           395M  1,1M  394M   1% /run
/dev/sda2        20G   14G  5,5G  71% /
tmpfs           2,0G  124K  2,0G   1% /dev/shm
tmpfs           5,0M     0  5,0M   0% /run/lock
tmpfs           2,0G     0  2,0G   0% /sys/fs/cgroup
/dev/loop0       92M   92M     0 100% /snap/core/8689
/dev/loop1       92M   92M     0 100% /snap/core/8592
tmpfs           395M     0  395M   0% /run/user/1000
```

Redémarrer le groupe de serveur “webservers” :

```
ansible webservers -b -a "reboot"
```

A ce stade de l'exercice, nous ne sommes pas encore en mesure d'installer un package à distance. Pour cela, nous devons encore avancer dans la configuration de notre serveur Ansible.

Cependant, je pense que vous avez tous compris l'intérêt d'une telle application ! La gestion d'un parc de serveurs ou de postes de travail est largement facilité.

6. Utilisation des playbooks

Pour déployer des applications ou des fichiers de configuration, il est plus aisé d'utiliser des playbooks. Un playbook, pour faire simple, est un fichier qui va automatiser des tâches de manière séquentielle et/ou conditionnelle (par exemple, installer un programme, mais ne l'installer que s'il n'est pas déjà installé).

6.1. Création d'un playbook

Créons un nouveau fichier /etc/ansible/roles/update_common.yml

```
sudo vim /etc/ansible/roles/update_common.yml
```

Et insérez le code suivant :

```
---
- hosts: webservers
  become: yes
  become_user: root
  become_method: sudo

  tasks:
  - name: Installation des paquets communs à tous les serveurs
    apt:
      update_cache: yes
      state: latest
      name: ['glances', 'pwgen', 'sudo', 'tar', 'unzip', 'vim', 'whois']
```

Lors de la création du fichier, vous avez peut être constaté que :

- un rôle commence toujours par ---,
- hosts : webservers est directement dans le fichier et non plus dans la commande,
- on nomme ensuite nos tâches tasks avec name, cela permet d'avoir un libellé compréhensible durant l'exécution du rôle,
- update_cache=yes permet de d'exécuter la commande apt-get update
- state=latest permet de s'assurer que la version du paquet est bien la dernière version disponible.

Après l'avoir enregistré, vous pouvez vérifier que la syntaxe de votre fichier .yml est correcte via la ligne de commande suivante :

```
ansible-playbook --syntax-check /etc/ansible/roles/update_common.yml
```

Vous devez obtenir un résultat similaire à celui-ci :

```
roger@ubuntu-preprod:/etc/ansible$ ansible-playbook --syntax-check roles/update_common.yml
playbook: roles/update_common.yml
```

6.2. Exécution d'un playbook

Pour exécuter et installer la liste des paquets sur tous les serveurs il faut utiliser la commande suivante :

```
ansible-playbook -i hosts /etc/ansible/roles/update_common.yml
```

A l'issue de la commande vous devez obtenir un résultat similaire à celui-ci :

```
roger@ubuntu-preprod:/etc/ansible$ ansible-playbook -i hosts
/etc/ansible/roles//update_common.yml

PLAY [webservers]
*********************************************************************************************
***

TASK [Gathering Facts]
*********************************************************************************************
ok: [192.168.1.69]
ok: [127.0.0.1]

TASK [install common packages for all servers]
*************************************************************
changed: [192.168.1.69] => (item=[u'htop', u'pwgen', u'sudo', u'tar', u'unzip', u'vim',
u'whois'])
changed: [127.0.0.1] => (item=[u'htop', u'pwgen', u'sudo', u'tar', u'unzip', u'vim', u'whois'])

PLAY RECAP
*********************************************************************************************
**********
127.0.0.1                  : ok=2    changed=1    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
192.168.1.69               : ok=2    changed=1    unreachable=0    failed=0    skipped=0
rescued=0    ignored=0
```

De manière générale, sur le site officiel d'Ansible, il existe un guide des bonnes pratiques pour l'utilisation de cette application. N'hésitez pas à le consulter : Best Pratices

6.3. Les variables

Il est possible de définir manuellement des variables dans nos playbooks. Nos variables Ansible, au même titre que des variables dans un script bash, vont nous permettre de définir des valeurs de configuration pour les différentes tâches. Mais Ansible fournit également de nombreuses variables avec ces différents modules.

Dans Ansible, les variables peuvent être définies à différents endroits : dans les playbooks, dans les rôles, dans l'inventaire, dans des fichiers séparés, en ligne de commandes, etc. Cncernant les variables et leur organisation, il faut savoir qu'il y a une priorité, selon où se trouve la varaibel, cette notion de priorité est très bien détaillée dans la documentation d'Ansible : https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html

Prenons l'exemple des variables fournies par le module setup. celui-ci permet de collecter les données sur l'hôte distant :

```
ubuntu@ubuntu-preprod:/etc/ansible$ ansible webservers -m setup
192.168.1.89 | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "192.168.1.89"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::a00:27ff:feb7:e2b0"
        ],
        "ansible_apparmor": {
            "status": "enabled"
        },
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "12/01/2006",
        "ansible_bios_version": "VirtualBox",
        "ansible_cmdline": {
            "BOOT_IMAGE": "/vmlinuz-5.4.0-66-generic",
            "maybe-ubiquity": true,
            "ro": true,
            "root": "/dev/mapper/ubuntu--vg-ubuntu--lv"
        },
        "ansible_date_time": {
            "date": "2021-03-03",
            "day": "03",
            "epoch": "1614784453",
            "hour": "15",
            "iso8601": "2021-03-03T15:14:13Z",
            "iso8601_basic": "20210303T151413803735",
            "iso8601_basic_short": "20210303T151413",
            "iso8601_micro": "2021-03-03T15:14:13.803798Z",
            "minute": "14",
            "month": "03",
            "second": "13",
            "time": "15:14:13",
            "tz": "UTC",
            "tz_offset": "+0000",
            "weekday": "Wednesday",
            "weekday_number": "3",
            "weeknumber": "09",
            "year": "2021"
        },
        "ansible_default_ipv4": {
            "address": "192.168.1.89",
            "alias": "enp0s3",
            "broadcast": "192.168.1.255",
            "gateway": "192.168.1.1",
            "interface": "enp0s3",
            "macaddress": "08:00:27:b7:e2:b0",
            "mtu": 1500,
            "netmask": "255.255.255.0",
```

```
            "network": "192.168.1.0",
            "type": "ether"
        },
        "ansible_default_ipv6": {},
        "ansible_device_links": {
            "ids": {
                "dm-0": [
                    "dm-name-ubuntu--vg-ubuntu--lv",

"dm-uuid-LVM-OWAOuUraPA3Z0tYBH5UeqSFlWC7xh6g91Vg7IqcV5G5ejEZyyvL4CJ6he92TlTHh"
                ],
                "sda": [
                    "ata-VBOX_HARDDISK_VB5147a87b-58191165",
                    "scsi-0ATA_VBOX_HARDDISK_VB5147a87b-58191165",
                    "scsi-1ATA_VBOX_HARDDISK_VB5147a87b-58191165",
                    "scsi-SATA_VBOX_HARDDISK_VB5147a87b-58191165"
                ],
                "sda1": [
                    "ata-VBOX_HARDDISK_VB5147a87b-58191165-part1",
                    "scsi-0ATA_VBOX_HARDDISK_VB5147a87b-58191165-part1",

[...]
```

Pour des raisons de longueur, l'exemple ci-dessus est volontairement coupé, mais comme vous pouvez le constater, le module setup nous renvoie de nombreuses variables système. Rien de plus simple pour les utiliser dans nos playbooks, il suffit de les appeler comme ceci :

```
# variable simple
{{ ansible_architecture }}

# variable pour accéder à une propriété
{{ ansible_date_time.date }}

# variable pour accéder à un tableau (première propriété)
{{ ansible_all_ipv4_addresses[0] }}
```

Mais les variables deviennent vraiment intéressantes quand on peut les définir de manière dynamique. Par exemple, lorsque nous souhaitons récupérer la liste des virtuals hosts installés sur un serveur pour les réutiliser plus tard :

```
ubuntu@ubuntu-preprod:/etc/ansible/roles$ cat test.yaml
---
- hosts: webservers
  become: yes
  become_user: root
  become_method: sudo

  tasks:
```

```
  - name: Lister les virtuals hosts
    command: /bin/ls /etc/apache2/sites-enabled/
    register: vhosts
```

La valeur vhosts contient la liste des éléments présents dans /etc/apache2/sites-enabled renvoyés par la commande ls.

Mais comment être sûr de ce qu'il y a dans nos variables ? Pour cela, nous utiliserons l'objet debug. En reprenant l'exemple ci-dessus :

```
ubuntu@ubuntu-preprod:/etc/ansible/roles$ cat test.yaml
```

```
---
- hosts: webservers
  become: yes
  become_user: root
  become_method: sudo

  tasks:
  - name: Lister les virtuals hosts
    command: /bin/ls /etc/apache2/sites-enabled/
    register: vhosts
  - debug: msg="{{ vhosts }}"
```

Tapez ensuite dans un terminal la commande pour tester votre playbook :

```
ubuntu@ubuntu-preprod:/etc/ansible/roles$ ansible-playbook test.yaml
```

Résultat de la commande :

```
PLAY [webservers]
*********************************************************************************************
*********************************************************************************

TASK [Gathering Facts]
*********************************************************************************************
*********************************************************************************
ok: [192.168.1.89]

TASK [Lister les virtuals hosts]
*********************************************************************************************
*********************************************************************
changed: [192.168.1.89]

TASK [debug]
*********************************************************************************************
*********************************************************************************************
```

```
ok: [192.168.1.89] => {
    "msg": {
        "changed": true,
        "cmd": [
            "/bin/ls",
            "/etc/apache2/sites-enabled/"
        ],
        "delta": "0:00:00.002328",
        "end": "2021-03-03 15:39:19.788748",
        "failed": false,
        "rc": 0,
        "start": "2021-03-03 15:39:19.786420",
        "stderr": "",
        "stderr_lines": [],
        "stdout": "000-default.conf",
        "stdout_lines": [
            "000-default.conf"
        ]
    }
}

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

Le résultat de la commande est relativement verbeux. Comme vu précédemment, pour récupérer la valeur de notre vhost 000-default-conf, il faudra appeler la variable {{ vhost.stdout_lines[0] }}.

6.4. Les handlers

Les handlers ressemblent aux tâches dans un playbook Ansible, mais ne sont exécutés que si la tâche contient la directive ***notify*** et que la tâche reçoit le signal que quelque chose a été changé. Par exemple, toujours avec notre serveur web, plutôt que de redémarrer Apache à chaque fois qu'une tâche est exécutée, alors, avec les handlers, le service redémarrera uniquement s'il y a eu un changement de configuration. Autre avantage, et pas des moindres, l'action en question ne sera lancée qu'après l'exécution de tous les blocs tâches.

7. Les rôles Ansible

Un playbook peut être monolithique ou divisé en plusieurs parties. Cela dépendra des tâches à réaliser, mais dès qu'on commence vraiment à mettre les mains dans les playbooks, qu'on a plusieurs tâches différentes, cela devient vite le bazar. C'est là qu'interviennent les rôles Ansible.

Les rôles définis par Ansible sont un ensemble de tâches qui s'assurent de la présence/absence d'une fonctionnalité spécifique (allant de la création d'un utilisateur Linux, en passant par l'installation et la configuration d'un serveur LAMP). Chaque rôle doit être capable de fonctionner en autonomie (sans compter les dépendances vers d'autres rôles) et inclut pour ça, dans son arborescence, un certain nombre de choses.

Ansible met à disposition un outil en CLI afin de préparer l'arborescence d'un rôle vide. Nous allons créer pour commencer le rôle common. Ce rôle sera notre base pour les serveurs.

```
ubuntu@ubuntu-preprod:/etc/ansible/roles$ sudo ansible-galaxy init common
- Role common was created successfully
```

La commande va générer une arborescence telle que :

```
ubuntu@ubuntu-preprod:/etc/ansible/roles$ tree common
common
├── README.md
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml

8 directories, 8 files
```

Nous avons donc :

- un fichier README.md pour documenter l'utilisation du rôle,
- un dossier defaults pour définir les variables par défaut,
- un dossier files où l'on mettra les fichiers hors playbooks dont on aura besoin (par exemple des fichiers de configuration),
- un dossier handlers qui recensera les événements Ansible liés à ce rôle,

- un dossier meta qui contient, qui contient comme son nom l'indique, les métadonnées de ce rôle telles que son auteur ou ses dépendances à d'autres rôles,
- un dossier tasks, le plus important, qui contient les tâches à exécuter,
- un dossier templates, qui contiendra les templates Jinja2 qui permettent de générer des fichiers textes de façon procédurale,
- le dossier vars pour définir les variables du rôle (qui supplante évidemment les définitions du dossier defaults).

L'utilisation des rôles Ansible semble donc parfaite pour s'assurer qu'une partie du code sera le plus générique possible et réutilisable, pour peu qu'on suive quelques principes de base.

Il n'y a pas d'obligation quant à l'utilisation des répertoires. S'ils sont vides, on peut les supprimer.