M3105 : Gestion Configuration d'un Parc de Serveurs avec Ansible

October 5, 2021

1 Installation Environnement

Créez deux serveurs Debian. L'un aura le rôle de serveur à configurer et l'autre de manager.

1. Connectez vous à la machine "manager et passez root
2. Editez le fichier /etc/hostname et remplacez "debian" par "manager"
3. Editez le fichier /etc/hosts et remplacez "debian" par "manager"
4. Redémarrez la machine :

```
[...]$ systemctl reboot
```

Répétez ces opérations sur la machine "server1".

Sur le manager, installez Ansible :

```
apt-get update
apt-get install ansible
```

Ansible accède aux serveurs via ssh. La configuration commune pour accéder à un serveur d'entreprise est la suivante : on ne peut pas se loger dessus en ssh en tant que root. Il faut se loger en tant qu'utilisateur normal (pas root) puis devenir root. On va mettre en place une connexion pour l'utilisateur rt par clé et non pas mot de passe. Sur le manager, créez une paire de clef publique/privée via la commande pour rt et en tant que rt **(ne travaillez pas en root!!!!)** :

```
ssh-keygen  (Ne pas mettre de mot de passe sur la clef)
```

Il faut maintenant mettre la clef publique dans la liste des clefs autorisées sur le serveur avec la commande : Attention : ces commandes s'effectuent depuis le manager (machine Ansible) et on envoie les clefs sur le serveur! Vérifiez que vous n'avez pas fait un ssh préalable sur le serveur (regardez l'adresse IP!)

```
ssh-copy-id -i ~/.ssh/id_rsa.pub rt@[adresse_ip_serveur]
```

Si tout se passe bien, vous devez pouvoir vous loger sur le serveur **sans** mot de passe :

```
ssh rt@[adresse_ip_serveur]
```

L'idée de cette authentification est que le serveur envoie une chaîne de caractères que le client (ici le manager) doit crypter avec la clef privée. Le client renvoie la chaîne cryptée et si le serveur arrive à décrypter avec la clef publique, c'est que le client possède bien la clef privée et il est authentifié et autorisé à accéder (tout cela sera approfondi en M3104), voir figure 1 (source : `https://www.digitalocean.com/community/tutorials/how-to-configure-ssh-key-based-authentication-on-a-linux-server`).

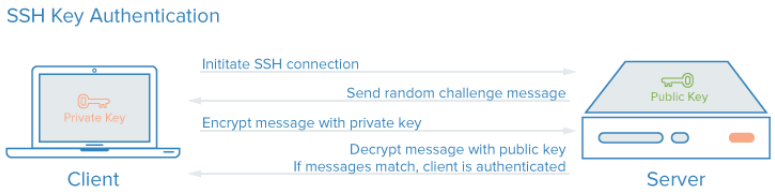

Figure 1: Clés SSH

Sur le manager, logez vous en rt et créez un répertoire playbook dans lequel nous allons placer toutes les recettes qu'Ansible utilisera pour configurer le serveur.
Il nous manque encore une dernière chose sur le serveur: faire en sorte que rt puisse faire un sudo sans mot de passe. Il faut créer le fichier /etc/sudoers.d/rt (installez le paquet sudo si ce répertoire manque) et le remplir comme suit :

```
rt ALL=(ALL) NOPASSWD: ALL
```

Vérifiez ensuite que tout fonctionne en vous re-logeant en tant que rt sur le serveur.

2 Connexion et Exécution d'un script simple

Ansible accède aux serveurs via ssh et cherche les paramètres de connexion aux serveurs à configurer dans le fichier ansible.cfg qui peut-être à divers endroits :

1. pointé par la variable d'environnement ANSIBLE_CONFIG
2. ansible.cfg (fichier local à l'endroit où on exécute ansible)
3. /.ansible.cfg si vous souhaitez avoir un seul fichier global.
4. /etc/ansible/ansible.cfg

Nous allons utiliser l'option 2 car souvent on met les recettes et les « coordonnées » des serveurs au même endroit (pour pouvoir ensuite tout sauvegarder d'un coup, par exemple via git ou svn). Créez donc un fichier ansible.cfg avec la configuration suivante :

```
[defaults]
inventory = hosts
remote_user = rt
private_key_file = ~rt/.ssh/id_rsa
fact_caching = jsonfile
fact_caching_connection = ./facts.d

[ssh_connection]
ssh_args = -o StrictHostKeyChecking=no
```

La première ligne indique que le fichier avec les noms des hôtes (l'inventaire) s'appelle hosts. Nous allons donc créer le fichier hosts dans notre répertoire playbook et y ajouter les coordonnées du serveur (**adapter** l'adresse IP à votre cas bien sûr) :

```
testserver ansible_ssh_host=192.168.1.27
```

«testserver» va servir d'alias dans les commandes ansible. On peut ensuite effectuer sa première commande ansible :

```
ansible testserver -m ping
```

« -m » introduit le module qui va être utilisé par ansible. Le module « ping » vérifie que le serveur est bien accessible, c'est-à-dire que la connectivité ssh est bonne. Si tout se passe bien, vous devriez obtenir :

```
testserver | success >> {
     "changed": false,
     "ping": "pong"
}
```

Si ça se passe mal, il faut débugger avec -vvv....
Testons un autre module : command. Ce module permet d'exécuter une commande arbitraire sur le serveur avec l'option -a :

```
 ansible testserver -m command -a uptime
```

Le module « command » est en fait le module par défaut et il est facultatif. Ré-exécutez la commande précédente en l'omettant. Certaines commandes nécessitent d'être root sur la machine distante. Testez les commandes

```
ansible testserver -m command -a "tail -v /var/log/syslog"
ansible testserver -b -m command -a "tail -v /var/log/syslog"
```

Que concluez-vous de l'analyse des résultats ?

On peut aussi installer des paquets avec le module « apt » (il ne s'agit pas de la commande apt directement, mais d'un module qui va utiliser apt et automatiser certaines actions comme la réponse automatique à la question « Do you want to continue? [Y/n] » qui apparaît lors de l'installation d'un paquet. Nous allons installer un serveur Web nginx. Avant de l'installer vérifiez si il n'y a pas déjà un serveur Web qui tourne sur la machine et, si c'est le cas, arrêtez le (1ère commande), retirez apache du démarrage automatique (2nde commande) puis dé-installez totalement le paquet :

```
apt purge apache2
apt-get autoremove
```

On peut maintenant installer nginx :

```
ansible testserver -b -m apt -a name=nginx
```

Vérifier que c'est bien nginx qui est là
Ce n'est pas en général une bonne idée d'installer directement un paquet. Il faut commencer par mettre à jour la base des paquets (la télécharger du dépôt) de manière à demander une version qui existe encore d'un paquet. On peut le faire faire à Ansible au travers de l'ajout de l'option update_cache après le nom du paquet :

```
ansible testserver -b -m apt -a "name=nginx update_cache=yes"
```

On peut aussi avoir besoin de redémarrer un service :

```
ansible testserver -b -m service -a "name=nginx  state=restarted"
```

Avant de faire les choses plus en grand (configuration complète, plusieurs serveurs), on va régler un dernier petit détail de sécurité en cryptant la clé privée. En effet, si quelqu'un la lit, il peut se connecter au serveur sans problème. On va donc la chiffrer. Le problème dans ce cas c'est qu'il va falloir la décrypter à chaque fois qu'on l'utilise... pas pratique. On va régler ce problème avec l'utilitaire ssh-agent qui garde en mémoire les mots de passe des clefs privées pour nous.
On crypte la clef sur le manager – en se plaçant dans le répertoire où est cette clef !! - avec un mot de passe de 4 lettres minimum :

```
ssh-keygen -p -f id_rsa
```

Manipulation côté manager :

1. On démarre ssh-agent ainsi :

   ```
   eval $(ssh-agent)
   ```

2. On ajoute la clef privée à ssh-agent en faisant un

   ```
    ssh-add ~rt/.ssh/id_rsa
   ```

   Si l'agent demande le mot de passe, c'est que c'est gagné....

3. On vérifie les clef connues par l'agent avec

   ```
   ssh-add -L
   ```

3 Premier playbook

Un playbook est un script Ansible qui permet d'enchainer plusieurs actions pour mener à la configuration complète d'un service sur une machine. Prenons un premier exemple simple qui montre comment on passe de la ligne de commande au script. On va mettre dans ce script l'équivalent des 2 commandes que l'on appellera tâches :

```
ansible testserver -b -m apt -a name="nginx update_cache=yes"

ansible testserver -b -m service -a "name=nginx  state=restarted"
```

Créez un fichier nginx.yml dans le répertoire playbook qui contient :

```
- name: Configure webserver with nginx
  hosts: testserver
  become: yes
  tasks:
    - name: install nginx
      apt: name=nginx update_cache=yes
      notify: restart nginx
  handlers:
    - name: restart nginx
      service: name=nginx state=restarted
```

become: yes permet de passer root sur le serveur.
On va jouer ce playbook avec la commande :

```
ansible-playbook nginx.yml
```

Si tout se passe bien, vous devriez avoir une sortie du type de la figure 2.

```
vagrant@ansible:~/playbook$ ansible-playbook nginx.yml

PLAY [Configure webserver with nginx] ************************************

GATHERING FACTS ***************************************************************
ok: [testserver]

TASK: [install nginx] *********************************************************
changed: [testserver]

TASK: [restart nginx] *********************************************************
changed: [testserver]

PLAY RECAP ********************************************************************
testserver                 : ok=3    changed=2    unreachable=0    failed=0
```

Figure 2: Clés SSH

On peut lister les tâches qui sont contenues dans un playbook et en avoir une explication synthétique avec la commande

```
ansible-playbook  --list-tasks nginx.yml
```

De la même manière, on peut lister les machines que ce playbook va modifier

```
ansible-playbook  --list-hosts nginx.yml
```

3.1 Facts

Si on regarde la sortie de « ansible-playbook nginx.yml » on voit qu'en fait il y a une tâche de plus que ce nous donne l'option « –list-tasks ». Il s'agit de la phase de collecte d'informations (Facts Gathering) sur le serveur distant. Ces informations sont stockées par Ansible et utilisables par la suite dans le script si besoin. On peut voir l'ensemble des informations collectées avec le module setup de la commande en ligne ansible :

```
ansible all -m setup
```

Testez cette commande et analysez les informations collectées pour le serveur testserver ? Comment sont-elles collectées par Ansible ?

3.2 Idempotence

Commencez cette partie en déinstallant avec nginx. Basez vous sur la documentation ansible pour le module apt (il faut un purge et un auto remove) `https://docs.ansible.com/ansible/latest/modules/apt_module.html`:

```
ansible testserver -b -m apt -a "name=nginx-common state=absent purge=yes autoremove=yes"
```

sur le serveur.

```
- name: Configure webserver with nginx
  hosts: testserver
  become: yes
  tasks:
    - name: install nginx
      apt: name=nginx update_cache=yes
      notify: restart nginx
  handlers:
    - name: restart nginx
      service: name=nginx state=restarted
```

Une propriété importante des playbooks que l'on souhaite avoir est l'idempotence. Cela veut dire que si l'on exécute plusieurs fois le même script, Ansible ne reporte que les modifications effectivement faites sur le serveur. Vérifiez ce qui se passe en exécutant plusieurs fois le playbook nginx.yml et en analysant la sortie retournée par ansible.

3.3 Un exemple plus complet de configuration de serveur web

On veut configurer de manière propre un serveur nginx. Commençons par retirer le paquet nginx de la machine testserver (apt-get purge nginx). Les fichiers de configuration ci-dessous (web.yml, templates/index.html.j2 et files/default) vont nous permettre de :

1. Installer nginx
2. Modifier un de ses fichiers de configuration, celui donnant le site par défaut. On va copier le fichier depuis le manager vers le testserver avec le module file.
3. Modifier la page d'accueil en la personnalisant avec une variable d'environnement d'Ansible. Il va falloir utiliser le module template qui permet de customiser un fichier.
4. Redémarrer nginx pour prendre en compte les modifications.

Voici les contenus des fichiers :
Fichier web.yml

```
- name: Configure webserver with nginx
  hosts: testserver
  become: yes
  tasks:
    - name: install nginx
      apt: name=nginx update_cache=yes
    - name: copy nginx config file
      copy: src=files/default dest=/etc/nginx/sites-available/default
    - name: enable configuration
      file: >
        dest=/etc/nginx/sites-enabled/default
        src=/etc/nginx/sites-available/default
        state=link
    - name: copy index.html
      template: src=templates/index.html.j2 dest=/usr/share/nginx/html/index.html mode=0644
    - name: restart nginx
      service: name=nginx state=restarted
```

Fichier default à placer dans le répertoire ∼/playbook/files à créer.

```
server {
listen 80 default_server;
listen [::]:80 default_server ipv6only=on;
root /usr/share/nginx/html;
index index.html index.htm;
server_name localhost;
```

```
location / {
try_files $uri $uri/ =404;
} }
```

Fichier index.html.j2 à placer dans le répertoire ∼/playbook/templates (l'extension j2 sert à identifier les templates pour Ansible) :

```
<html>
  <head>
<title>Welcome to ansible</title> </head>
<body>
<h1>nginx, configured by Ansible</h1>
<p>If you can see this, Ansible successfully installed nginx.</p>
<p> This server is running under {{ ansible_lsb.description }}</p>
  </body>
</html>
```

Analyser succinctement le playbook avec l'option « – list-taks ». Appliquez le template sur le serveur testserver. Observez le résultat sur le site Web. Modifier le template index.html.j2 de manière à ce que soit retournée dans la page Web l'adresse IP de l'interface eth1. Aidez-vous du module debug : cherchez ansible module debug sur Google!

3.4 Appliquer une recette à un ensemble de serveurs.

Le fichier host peut contenir plein de machines et pas une seule car bien sûr Ansible peut gérer plusieurs machines en parallèle. On peut également faire des groupes de machines.
Modifiez votre fichier host comme suit de manière à appliquer Ansible sur votre manager, en plus de la machine distante:

```
testserver ansible_ssh_host=10.x.y.z
testserver2 ansible_ssh_host=localhost
[webservers]
testserver
testserver2
```

On voit que l'on créée des alias (testserver et testserver2) puis que l'on créée des groupes qui peuvent être des groupes de serveurs, ici webserver. Faites les tests suivants :

```
ansible webservers  -a "uptime"
ansible all -a "uptime"
```

Bon, vous avez du voir un problème avec les clefs car le second serveur, localhost (la machine admin dans notre cas), n'a pas été configuré pour fonctionner avec Ansible. Il faut que :

- rt soit capable de faire un sudo sans mot de passe (comme dans la section 1)
- rt soit capable de faire un ssh avec clef publique.

Sur localhost, vous avez déjà, dans le répertoire /home/rt/.ssh une clef publique id_rsa.pub. Lorsqu'on cherche à se loger en ssh avec une clef, le serveur ssh cherche à lire la clef dans le fichier /home/rt/.ssh/authorized_keys. Il suffit donc de copier id_rsa.pub dans authorized_keys.
Appliquez à nouveau les commandes ansible et montrer que cela fonctionne sur les 2 serveurs.

4 Notion de Rôles

Les rôles sont un niveau d'abstraction supplémentaire dans Ansible. Définir des rôles permet de les combiner par la suite et de « normaliser » la présentation d'un playbook. Nous allons transformer notre playbook en rôle puis voir rapidement Galaxy, le hub d'Ansible où les utilisateurs publient leurs rôles.

4.1 Ecrire un rôle

Faisons un peu de ménage. On va à la racine de rt puis on créée un répertoire Ansible. Dans ce répertoire, on ajoute un répertoire roles qui va stoker les rôles. On créée ensuite un sous-répertoire nginx qui va contenir notre rôle nginx et différents sous répertoires :

```
mkdir roles
cd roles
mkdir nginx
cd nginx
mkdir files handlers meta templates tasks vars
```

On va ensuite dans tasks où nous allons mettre l'équivalent du web.yml du playbook.
Première simplification : dans tasks, il ne doit y avoir que les tâches et plus aucun en-tête. De plus, avec les rôles, on a plus besoin de donner les noms des répertoires pour les fichiers et les templates car ils sont implicites. Cela simplifie le fichier web.yml **qu'il faut renommer en main.yml**:

```
- name: install nginx
  apt: name=nginx update_cache=yes
- name: copy nginx config file
  copy: src=default dest=/etc/nginx/sites-available/default
- name: enable configuration
  file: >
        dest=/etc/nginx/sites-enabled/default
        src=/etc/nginx/sites-available/default
        state=link
- name: copy index.html
   template: src=index.html.j2 dest=/usr/share/nginx/html/index.html mode=0644
- name: restart nginx
  service: name=nginx state=restarted
```

Il faut ensuite copier les fichiers default et index.html.j2 dans les bons répertoires.
On se replace ensuite au-dessus du répertoire roles et on créée un fichier play.yml Ce fichier va utiliser le role nginx que l'on vient de créer comme suit :

```
- hosts: webservers
  become: yes
  roles:
    - role: nginx
```

Il faut aussi copier les fichiers ansible.cfg et hosts dans le répertoire Ansible.
On joue ensuite play.yml à l'aide de la commande

```
ansible-playbook play.yml
```

Si on avait défini plusieurs rôles correspondants à plusieurs étapes de configuration, on aurait pu simplement les ajouter au fichier play.yml par exemple comme suit :

```
- hosts: wordpress_hosts
  become: yes
  roles:
    - role: nginx
    - role: php
    - role: mysql
    - role: wordpress
```

4.2 Utiliser le hub d'Ansible

Les utilisateurs publient leur rôle sur `https://galaxy.ansible.com/` (menu « browse »). L'installation des rôles se fait dans le repertoire /etc/ansible. Il faut donc être root sur votre machine. Il faut aussi que le proxy http/https soit bien positionné. Si

```
env | grep http
```

ne retourne rien, alors il faut installer :

```
export https_proxy=http://134.59.136.251:3128
export http_proxy=http://134.59.136.251:3128
```

Placez vous dans votre répertoire roles. Sélectionnez des rôles pour nginx et installez les rôles sur votre machine à l'aide de la commande ansible-galaxy. (exemple avec geerlingguy.nginx)

```
 ansible-galaxy install --roles-path . geerlingguy.docker
```

Fouillez ensuite dans le répertoire correspondant.