M3105 : Supervision réseau et application Ansible

Introduction :

Ansible est un logiciel permettant de configurer un grand nombre de serveurs en poussant la configuration au travers de SSH. Aucune installation préalable n'est donc nécessaire sur le ou les serveurs, pour eux c'est transparent.

1. Installation Environnement

Création de clés ssh de manager vers le serveur et autorisation du sudo pour rt.

Pour ce TP nous allons travailler avec une machine qui jouera le rôle de manager et une autre qui jouera le rôle de serveur.
La première chose à faire est de générer une paire de clés SSH (sans mot de passe pour l'instant) sur le manager afin de pouvoir se connecter sur le serveur sans avoir à taper de mot de passe.
Pour ça :

#ssh-keygen

```
ky002500@Lofi:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ky002500/.ssh/id_rsa): ansible
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in ansible
Your public key has been saved in ansible.pub
The key fingerprint is:
SHA256:FXwGJ63brgwkLO92RzAaGCoHiloY0gjvVR+fZdlCJu8 ky002500@Lofi
The key's randomart image is:
+---[RSA 3072]----+
|oo    . ..=+*o   |
|=.. .. . o.@* .  |
|++...o  . ++..   |
|=.+.... o...     |
|.+. . ooSo oE    |
|.    o.o  o .    |
|      . .. .     |
|     .. .o. .    |
|     ... .o.     |
+----[SHA256]-----+
ky002500@Lofi:~$
```

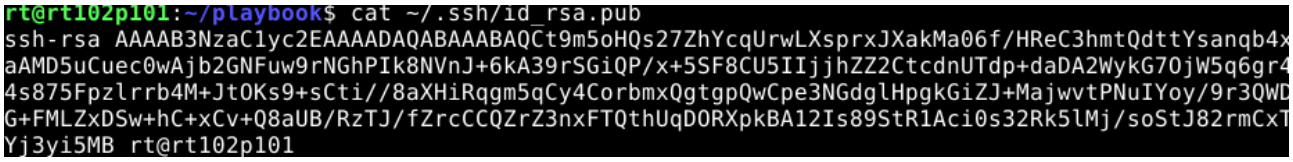
Exemple de génération d'une paire de clés SSH

Maintenant que les clés sont générées il faut déposer la clé publique sur le serveur.

Pour ça :

#ssh-copy-id rt@10.0.1.101

Il faudra tout de même taper le mot de passe pour copier la clé, mais après cela, plus besoin de taper le mot de passe lorsqu'on fait un ssh vers ce serveur.

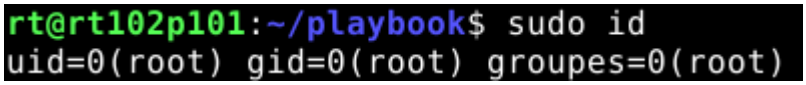


La clé ssh publique à bien été déposé sur le serveur

Il manque une dernière chose avant de pouvoir utiliser Ansible, c'est le fait de pouvoir fait un sudo sans mot de passe sur le serveur, depuis le manager. Pour ça, c'est très simple. On se connecte en ssh au serveur, on crée le fichier /etc/sudoers.d/rt et on le rempli avec la ligne suivante :

rt ALL=(ALL) NOPASSWD: ALL

On doit obtenir ceci en rt sur le serveur :



Vérification de l'absence de mot de passe pour sudo

Maintenant que nous sommes prêts on peut installer Ansible sur le manager

#apt update

#apt install ansible

2. Connexion et Exécution d'un script simple

Toujours sur le manager on crée un répertoire playbook.
Dans ce répertoire on va commencer par crée notre fichier de configuration ansible.cfg et le remplir de la sorte :

```
[defaults]
hostfile = 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
```

Contenu du fichier de configuration ansible.cfg

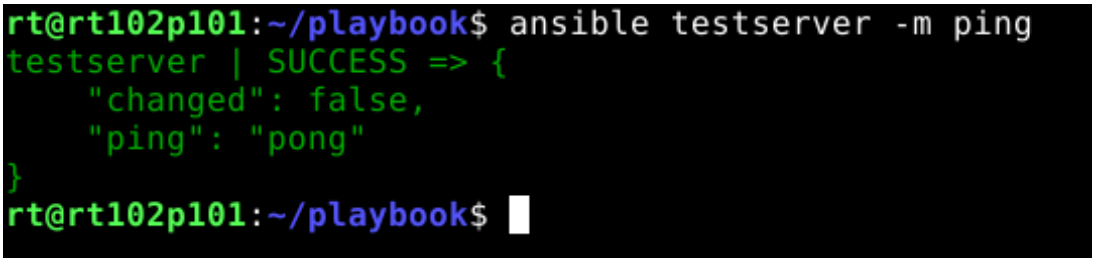
La première ligne indique que le fichier avec les noms des hôtes s'appelle hosts. Nous allons donc le créer dans notre répertoire playbook et y ajouter les coordonnées du serveur de la sorte :
testserver ansible_ssh_host=10.0.1.102

testserver va devenir l'alias du serveur 10.0.1.102

On peut désormais effectuer des commandes Ansible tel que :

#ansible testserver -m ping

Qui permet de vérifier que le tunnel ssh fonctionne entre le manager et la machine.
« -m » introduit le module qui va être utilisé par ansible. Tout marche bien donc on obtient ça :

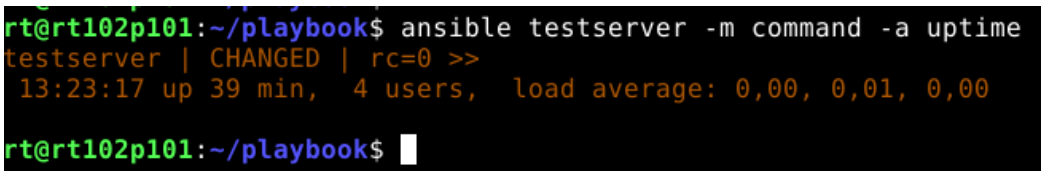


Test de la connexion au serveur depuis le manager grâce au module ping

Le module command permet d'exécuter une commande arbitraire sur le serveur avec l'option -a tel que :

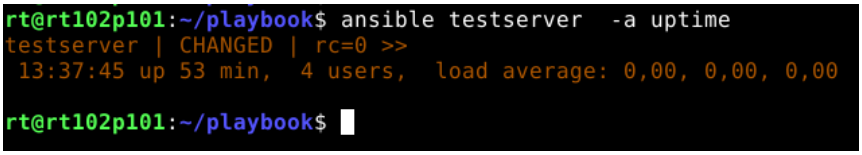
#ansible testserver -m command -a uptime

va exécuter uptime sur le serveur et retourner le résultat de la commande.



Test d'une commande avec le module command

Le module « command » est en fait le module par défaut et il est facultatif.
On peut donc l'omettre et obtenir le même résultat.



Test d'une commande sans le module command

Si une commande nécessite d'être root pour l'exécuter il faut ajouter l'argument -b, sinon ansible renvoie une erreur en nous précisant qu'il faut plus de droit. Par exemple :

```
rt@rt102p101:~/playbook$ ansible testserver -m command -a "tail -v /var/log/syslog"
testserver | FAILED | rc=1 >>
tail: impossible d'ouvrir '/var/log/syslog' en lecture: Permission non accordéenon-zero retur
n code

rt@rt102p101:~/playbook$
rt@rt102p101:~/playbook$
rt@rt102p101:~/playbook$ ansible testserver -b -m command -a "tail -v /var/log/syslog"
testserver | CHANGED | rc=0 >>
==> /var/log/syslog <==
Oct  8 13:39:18 rt102p111 systemd[1]: Started Session 233 of user rt.
Oct  8 13:39:18 rt102p111 systemd[1]: session-233.scope: Succeeded.
Oct  8 13:39:18 rt102p111 systemd[1]: Started Session 234 of user rt.
Oct  8 13:39:18 rt102p111 systemd[1]: session-234.scope: Succeeded.
Oct  8 13:39:18 rt102p111 systemd[1]: Started Session 235 of user rt.
Oct  8 13:39:18 rt102p111 systemd[1]: session-235.scope: Succeeded.
Oct  8 13:39:19 rt102p111 systemd[1]: Started Session 236 of user rt.
Oct  8 13:39:19 rt102p111 systemd[1]: session-236.scope: Succeeded.
Oct  8 13:39:19 rt102p111 systemd[1]: Started Session 237 of user rt.
Oct  8 13:39:19 rt102p111 ansible-command: Invoked with creates=None executable=None _uses_sh
ell=False _raw_params=tail -v /var/log/syslog removes=None argv=None warn=True chdir=None std
in=None

rt@rt102p101:~/playbook$
```

Test d'une commande ansible sans puis avec l'argument -b

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 on vérifie s'il n'y a pas déjà un serveur Web qui tourne sur la machine et, si c'est le cas, on le désinstalle totalement.
Sur le serveur :

#Apt purge apache2

#Apt autoremove

On peut maintenant installer nginx :

#ansible testserver -s -m apt -a "name=nginx update_cache=yes"

update_cache=yes permet de faire un apt update avant d'installer nginx afin de demander des version de paquet qui existe encore.

Pour démarrer le service on fait :

#ansible testserver -s -m service -a "name=nginx state=restarted"

Avant de continuer et de faire les choses en grand, on va améliorer la sécurité en chiffrant la clé shh. Pour ce faire on se place dans le répertoire de la clé ~/.ssh/ et on fait :

#ssh-keygen -p -f id_rsa

Le -p indique que l'on veut appliquer un mot de passe à cette clé et le -f id_rsa indique le nom de la clé que l'on veut chiffrer.

Quand on crée une clé chiffrée, c'est bien parce que ça ajoute de la sécurité mais c'est très embêtant car il faut taper la phrase de passe de la clé à chaque création de tunnel ssh. Donc pour une commande ansible sur 500 machines il faudra au moins taper 500 fois la même phrase de passe. Le bon compromis étant d'utiliser un agent ssh grâce à ssh-agent, de lui ajouter la clé chiffrée via ssh-add de taper une fois le mot de passe et de ne plus avoir à le taper pour la session en cours. La clé ssh en clair est alors stocké temporairement dans la RAM, il est donc quasi impossible pour un hacker de la récupérer et de compromettre l'ensemble des serveurs. Alors que si la clé est stockée en clair sans phrase de passe et qu'elle est récupérée, l'ensemble des serveurs peuvent être compromis.

On démarre donc l'agent avec :

#eval $(ssh-agent)

On ajoute la clé privée à l'agent ssh :

#ssh-add

On tape le mot de passe de la clé une seule fois ici.

Pour vérifier que notre clé ssh à bien été apprise par l'agent on tape

#ssh-add -L

L'agent nous afficher alors les clés publiques des clés privées qu'il connait. (Il ne va pas afficher la clé privée déchiffrée, ça serait une faille de sécurité)

```
rt@rt102p101:~/playbook$ ssh-add -L
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDGHYEs3M1PlDlH3MPVUXW5R6tJsDYTdslwiLfkVeboUVTykRq3AJ9XZ
b+o9NC84hlgnPruUaU3OjhCrczPQaOFOIZEn1P5YUEPCnUJ6s3cd9rJsbz77BRMOeiDnRJvLGNItODQICkV11ohdW5n/C
POdeA44CsqyU3dXGt55/Z2InkJk+Khtwh5KJBMyfJePWJDZZ6HB5ChcjKrS/2FepihXlTcwqdVPN2Khm75uLHzKeuBBAT
UGKH3g8cBiTK+1oXV5rknkZcjQK/NPl2MXMQzxFYLbuaxLMZplaEPnm/ZDdNMgZb0l84wo6fPeyVdl9jeXJdu86h5yifD
uIN7gJvz rt@rt102p101
```

L'agent retient actuellement une clé privée ssh

Nous sommes donc enfin prêts pour utiliser Ansible à grande échelle et de manière sécurisé.

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 -s -m apt -a name="nginx update_cache=yes"

#ansible testserver -s -m service -a "name=nginx state=restarted"

Pour ce faire on crée un fichier nginx.yml toujours dans playbook avec à l'interieur :

```
- 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
```

Contenu de nginx.yml (become : yes permet d'obtenir les droits admin)

On lance le playbook sur le serveur avec :

 # ansible-playbook nginx.yml

Et on obtient :

```
rt@rt102p101:~/playbook$ ansible-playbook nginx.yml

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

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

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

RUNNING HANDLER [restart nginx] ************************************************
changed: [testserver]

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

rt@rt102p101:~/playbook$
```

Résultat du playbook nginx.yml

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 que 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

Cette ligne donne plein d'information sur le serveur comme on peut le voir ci-après :

Début du résultat de la commande # ansible all -m setup

Pour récupérer ces informations, ansible exécute des commandes Linux sur le serveur comme ip ou date.

3.2 Idempotence

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. Par exemple si l'on demande à Ansible d'installer nginx et de le redémarrer il ne doit pas le redémarrer si nginx était déjà installé avant le lancement du script (on ne veut pas redémarrer un serveur pour rien). Autre exemple plus parlant : si je souhaite rendre un nombre positif, je peux faire simplement abs() de ce nombre mais si je nombre est déjà positif j'applique quand même la fonction (pour rien). Il faut donc vérifier avant si le nombre est négatif et donc s'il a besoin qu'on lui applique la fonction.
Typiquement dans notre script nginx.yml, c'est ce qu'on a fait.

```
- 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
```

Contenu du script nginix.yml

La partie handlers s'applique uniquement si la partie install a été un succès. Et la partie install nginx est un succès si et seulement si nginx n'est pas déjà installé sur le serveur.
Conclusion : si ngnix est déjà installé, il ne va pas être redémarré. Il y a idempotence.

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 personnaliser un fichier.
4. Redémarrer nginx pour prendre en compte les modifications.

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
```

Contenu du fichier web.yml

Fichier default à placer dans le répertoire ~/playbook/files

```
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;
} }
```

Contenu du fichier default (fichier de configuration qui va être envoyé sur le serveur par Ansible)

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>
```

Contenu du fichier index.html.j2

Le fichier index.html.j2 va être interprété par ansible et la variable vont recevoir sa valeur puis il va être envoyé sur le serveur.
ansible_lsb.description est la variable qui indique la version de l'OS (du serveur)
Si tout se passe bien on va donc se retrouver avec un page web accessible sur le port 80 du serveur avec la version de l'os (ansible ayant remplacé {{ ansible_lsb.description }} par la version de l'OS du serveur). Si l'on veut l'adresse ip de l'interface eth1 cela doit être possible dans la partie ansible_default_ipv4.

```
rt@rt102p101:~/playbook$ ansible-playbook web.yml

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

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

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

TASK [push config] **********************************************************
ok: [testserver]

TASK [copy config] **********************************************************
ok: [testserver]

TASK [push html] ************************************************************
ok: [testserver]

PLAY RECAP ******************************************************************
testserver                 : ok=5    changed=0    unreachable=0    failed=0

rt@rt102p101:~/playbook$
```

Résultat du playbook ansible web.yml

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.
On va ici ajouter la machine du manager au fichier host et crée un groupe « webservers » composé de ces deux machines.

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

Configuration du fichier host (testserver à pour IP 10.0.101.102)

Avant de pouvoir tester des commandes il faut configurer les clés de testserver2 (la machine du manager) Pour ça il suffit de copier la clé publique « id_rsa.pub » que l'on a créée au début du TP dans le fichier /home/rt/.ssh/authorized_keys.

On peut donc tester :

#ansible webservers -a "uptime"

#ansible all -a "uptime"

On remarque que ça marche et que les résultats sont identiques car le groupe webservers contient tous les serveurs déclarés (le manager et le serveur).