Atelier ANSIBLE Starinux

1. Prérequis docker pour l'atelier ansible

```
# apt install docker.io git -y
# adduser starinux
# adduser starinux docker
# su - starinux
$ cd
$ git clone https://github.com/patgg/coursansible.git
$ mv coursansible/ansible ansible
$ mv coursansible/docker docker
$ mv coursansible/lancement.yml ansible/
$ cd docker/SSHD/
$ docker network create starinux -d host --subnet=10.10.10.0/24 br0
$ docker build -t starinuxsshd-image .
$ docker run -d --name starinuxsshd -p 22004:22 --network=starinux starinuxsshd-image
$ docker start starinuxsshd
$ docker exec -ti starinuxsshd bash
```

← (si ça se connecte c'est que le container est fonctionnel ; faire exit pour en sortir et passer à l'étape suivante)

```
$ cd ../ANSIBLE/
$ docker build -t starinuxansible-image .
$ docker run -d --name starinuxansible -p 22005:22 -v /home/starinux/ansible:/home/starinux/ansible --network=starinux starinuxansible-image
$ docker start starinuxansible
$ docker exec -ti starinuxansible bash
```

← (si ça se connecte c'est que le container est fonctionnel ; faire exit pour en sortir et passer à l'étape suivante)

```
$
$ exit
# ln -s /home/starinux/ansible/inventaire /etc/ansible/inventaire
# su - starinux
$ cd ansible
```

2. Configurations de base

A) Créer un fichier ansible.cfg

```
$ ansible-config init --disabled -t all > /home/starinux/ansible/ansible.cfg
```

(init disponible sur les dernières versions)

```
# ln -s /home/starinux/ansible/ansible.cfg /etc/ansible/ansible.cfg
```

Lister les variables de la configuration :

```
$ ansible-config list
```

B) Quelques variables utiles à définir

```
inventory = /etc/ansible/inventaire/
```

← dossier d'inventaire général

```
ansible_home = ~/.ansible
```

← dossier .ansible sur l'hôte distant

```
become = True
```

← pour avoir les privilèges de superutilisateur par défaut

---

become_method = sudo
become_user = starinux ← utilisateur qui aura les droits sudo (il doit être dans le groupe sudo ou wheel sur l'hôte distant)
private_key_file = /home/starinux/.ssh/secret_key_rsa ← créer la clef
remote_user = starinux ← utilisateur sur l'appareil distant (vérifier qu'il existe!)
vault_password_file = /home/starinux/.vault_password.txt ← fichier local où se trouve le fichier de mots de passe
host_key_checking = False

Pour d'autres variables, aller voir :
https://docs.ansible.com/ansible/latest/reference_appendices/config.html

1. **Les outils ansible :**

Il existe plusieurs outils en ligne de commande dans l'outil ansible :

A) La commande ansible

la commande ansible permet d'agir sur les machines distantes sans playbook, de tester les modules et de voir la réponse des cibles.

ça utilise le fichier ansible.cfg et l'inventaire qui y est défini par défaut.

- Déposer au préalable la clef publique préalablement créé dans le fichier authorized_keys des machines cibles.

```
$ ssh-keygen -t rsa
$ ssh-copy-id -i ~/.ssh/secret_key_rsa.pub ABC@ploum
```

- Tester la connexion ou un module (ici shell).

```
$ ansible toust -m shell -a "ip a"
```

- Lister les facts disponibles sur la cible.

```
$ ansible localhost -m setup -a 'filter=ansible_*'
```

C'est d'un usage simple (« toust »/localhost est la machine cible, le « -m » est le module utilisé, le « -a » est l'action demandé). Pour modifier la source d'inventaire, vous pouvez utiliser l'option « -i » et y indiquer le fichier/dossier d'inventaire à utiliser.

Pour plus d'infos, aller voir :
https://docs.ansible.com/ansible/latest/command_guide/cheatsheet.html

B) La commande ansible-doc

Elle permet de lister tous les modules types disponibles :

```
$ ansible-doc -l
```

Les modules sont divisés par les types suivant :
become , cache, callback, cliconf, connection, httpapi, inventory, lookup, netconf, shell, vars, module, strategy

- Lister tous les modules disponibles pour le type inventory :

```
$ ansible-doc -t inventory -l
```

- Lister tous les modules disponibles pour le type connection :

```
$ ansible-doc -t connection -l
```

Pour plus d'informations, aller voir : https://docs.ansible.com/ansible/latest/cli/ansible-doc.html

C) La commande ansible-inventory

Elle permet de lister graphiquement l'inventaire du dossier (ou fichier) d'inventaire :

```
$ ansible-inventory -i /etc/ansible/inventaire --graph
```

Pour plus d'informations, aller voir : https://docs.ansible.com/ansible/latest/cli/ansible-inventory.html

D) La commande ansible-vault

Elle permet de chiffrer/déchiffrer des variables sensibles tels que les mots de passe.

- Créer un fichier mdp.yml avec la variable voulue :

```
mdp_starinux: associationstarinux
```

- Le chiffrer (indiquer le mot de passe 2 fois!) :

```
$ ansible-vault encrypt mdp.yml
```

- Voir la variable chiffrée (y indiquer le mot de passe) :

```
$ ansible-vault view mdp.yml
```

Pour plus d'informations, aller voir : https://docs.ansible.com/ansible/latest/cli/ansible-vault.html

E) La commande ansible-playbook

Elle permet de lancer un playbook contenant des taches ou des rôles.

```
$ ansible-playbook -i inventaire playbook.yml
```

Pour plus d'informations, aller voir : https://docs.ansible.com/ansible/latest/cli/ansible-playbook.html

F) La commande ansible-galaxy

Elle permet de récupérer des roles préalablement configurés (voir la partie roles) à partir d'un hub ansible (du style dockerhub, github, ...).

```
$ ansible-galaxy install geerlingguy.mysql
$ ansible-galaxy search elasticsearch
$ ansible-galaxy init starinux
```

La structuration en dossiers/fichiers d'un rôle est globalement le suivant :

```
roles/              ← dossier de base où on pose les rôles
  starinux/         ← dossier du rôle de starinux (où se trouve le reste)
    tasks/          ← dossier des playbook à lancer
      main.yml      ← fichier playbook de base
    handlers/       ← dossier des redémarreurs de services
      main.yml      ← fichier playbook de base
    templates/      ← dossier des modèles
      ntp.conf.j2   ← fichier playbook de base (ici fichier de config de ntp)
    files/          ← dossier des fichiers
      bar.txt       ← fichier
      foo.sh        ← fichier
    vars/           ← dossier des variables
      main.yml      ← fichier playbook de base
    defaults/       ← dossier des variables par default
      main.yml      ← fichier playbook de base <-- priorité basse
    meta/           ← dossier des meta
      main.yml      ← fichier playbook de base <-- role dependencies
```

Pour plus d'informations, aller voir : https://docs.ansible.com/ansible/latest/cli/ansible-galaxy.html

2. L'inventaire pour ansible

A) Le dossier d'inventaire

Pour avoir un inventaire facilement modifiable et adapté dans le cas de l'usage du format YAML, on créé le dossier « inventaire » (qui correspond à la variable indiqué dans ansible.cfg) dans */etc/ansible/* .
Puis on dépose les fichiers INI ou YAML dedans (voir ci dessous pour les deux formats d'inventaires utilisés).

Dans ce dossier, pour l'inventaire au format YAML, on créé deux dossiers 'group_vars' et 'host_vars' :

- Le premier est pour les variables de groupes : « test » dans notre exemple ci dessous, ce qui implique un fichier nommé test.yml dans le dossier group_vars.
- Dans le second dossier host_vars, c'est pour un appareil cible : « Nom1 » pour exemple ci dessous, ce qui implique un fichier nommé Nom1.yml.

B) Inventaire au format INI

```
localhost ansible_connection=local
srv23 ansible_host=IPNOMsrv23 ansible_user=Administrateur ansible_password=admin ansible_connection=winrm ansible_port=5986 ansible_winrm_server_cert_validation=ignore
[test]
Nom1 ansible_host=IPNOM1  <-- clef asymetrique
```

```
Nom2 ansible_host=IPNOM2 ansible_password=AZERTY ansible_connection=ssh <-- mot de passe à éviter -> preferer vault
[toust]
ploum ansible_host=IP25 ansible_user=ABC ansible_connection=ssh
[docker2]
dock2 ansible_host=IPdock2 ansible_user=root ansible_connection=container
[test:vars]
ansible_user=XYZ
ansible_connection=ssh
```

C) Inventaire au format YAML

```
all:
  children:
    test:
      hosts:
        Nom1:
          ansible_host=IPNOM1
        Nom2:
          ansible_host=IPNOM2
          ansible_password=AZERTY
    toust:
      hosts:
        ploum:
          ansible_host=IP25
    docker2:
      hosts:
        dock2:
          ansible_connection: container
    ungrouped:
      hosts:
        localhost:
          ansible_connection: local
        srv23
          ansible_host=IPNOMsrv23
```

- dans le dossier group_vars, un fichier test.yml :

```
ansible_connection: ssh
ansible_ssh_user: XYZ
```

- Dans le dossier host_vars, vous pouvez aussi, par exemple, affiner l'inventaire et créer un fichier ploum.yml :

```
ansible_connection: ssh
ansible_ssh_user: ABC
```

- et également un fichier pour le serveur srv23.yml :

```
ansible_user=Administrateur
```

```
ansible_connection=winrm
ansible_password=admin
ansible_port=5986
ansible_winrm_server_cert_validation=ignore
```

Pour de plus amples informations sur l'inventaire, voir :
https://docs.ansible.com/ansible/latest/inventory_guide/intro_inventory.html

3. Les structures des dossiers et fichiers pour ansible

La structuration d'un playbook est composé au minimum d'un fichier .yaml avec des instructions simples (composé de pleins de taches dans tasks) :

```
- hosts: test, toust
  tasks:
  - name:  tache 1 > poser le nom de la machine dans un fichier /tmp/toto
    shell: echo '{{ ansible_hostname }}' | tee /tmp/toto
```

hosts: ← définir un ou plusieurs groupes/hôtes cibles dans l'inventaire
tasks: ← classe de groupe de modules nécessaires à l'objectif du playbook
name: ← description de ce qui va être fait avec ce module ci
shell: ← module shell qui permet d'exécuter des commandes shell, ici ça ajoute la variable du nom d'inventaire dans le fichier toto se trouvant dans /tmp

Ça peut être composé d'un fichier d'entrée qui va appeler (par des includes_* dynamiques // import_* statiques) d'autres playbooks (composé de plusieurs tâches) :

```
- hosts: test, toust
  tasks:
  - name:  pour inclure un fichier playbook2
    include_tasks: playbook2.yml
```

Mais la structuration la plus adéquat, dans le temps, est l'utilisation de rôles. Un dossier de dossiers spécifique (une liste normé de dossiers : tasks, roles, vars, templates, defaults, handlers) définit un rôle. Il y a plusieurs manières et variations pour appeler le rôle :

```
- hosts: test, toust
  tasks:
  - name: Inclure le role starinux
    include_role:
      name: starinux
```

```
- hosts: test, toust
  roles:
    - starinux
    - role: starinux2
      vars :
        toto : vv
        titi : tt
```

Comme vu en 1.F/, il est possible de créer un rôle (ex : starinux), avec dossiers et fichiers par defaut, en faisant : ansible-galaxy init starinux

* modules

On peut lister tous les modules disponibles pour le type module :

```
$ ansible-doc -t module -l
```

On peut voir énormément de modules spécifiquement réseau ou systèmes, ou spécifiques à certains logiciels. Ces modules vont nous servir à structurer les requêtes des playbook ou des rôles.

4. <u>création d'un rôle</u>

on ne créera pas de playbook, mais un rôle avec les divers fichiers yaml de base avec leur description :

```
roles/        ← dossier de base où on pose les rôles
  starinux/    ← dossier du rôle de starinux (où se trouve le reste)
    tasks/         ← dossier des playbook à lancer
      main.yml     ← fichier playbook de base
    handlers/      ← dossier des redémarreurs de services
      main.yml     ← fichier playbook de base
    templates/     ← dossier des modèles
      ntp.conf.j2  ← fichier playbook de base (ici fichier de config de ntp)
    files/         ← dossier des fichiers
      bar.txt      ← fichier
      foo.sh       ← fichier
    vars/          ← dossier des variables
      main.yml     ← fichier playbook de base
    defaults/      ← dossier des variables par default
      main.yml     ← fichier playbook de base <-- priorité basse
    meta/          ← dossier des meta
      main.yml     ← fichier playbook de base <-- role dependencies
```

On a donc les taches, les handlers, les templates, les fichiers, les variables, les variables par défaut, et les meta données.