

Tâches Ansible Ad-Hoc Linux

Objectifs de certification

RHCE EX294 (RHEL8)

Si vous poursuivez des objectifs de certification voici ceux qui sont suggérés ici :

- **2. Maîtrise des composants de base d'Ansible**
 - 2.1. Inventaires
 - 2.2. Modules
 - 2.3. Variables
 - 2.7. Fichiers de configuration
 - 2.8. Utiliser la documentation fournie pour trouver des informations spécifiques aux modules et commandes Ansible
- **3. Installation et configuration d'un nœud de contrôle Ansible**
 - 3.1. Installer les paquets requis
 - 3.2. Créer un fichier d'inventaire statique des hôtes
 - 3.3. Créer un fichier de configuration
 - 3.4. Créer et utiliser des inventaires statiques pour définir des groupes d'hôtes
 - 3.5. Gérer les parallélismes
- **4. Configuration des nœuds gérés par Ansible**
 - 4.1. Créer et distribuer des clés SSH aux nœuds gérés
 - 4.2. Configurer la réaffectation des privilèges sur les nœuds gérés
 - 4.3. Valider une configuration fonctionnelle à l'aide des commandes Ansible ad hoc
- **5. Écriture de scripts pour les tâches d'administration**
 - 5.1. Créer des scripts shell simples
 - 5.2. Créer des scripts shell simples qui exécutent les commandes Ansible ad hoc
- **6. Création des jeux et des playbooks Ansible**
 - 6.1. Utiliser des modules Ansible courants
- **7. Utilisation des modules Ansible**
 - 7.1. Paquets logiciels et repos
 - 7.2. Services
 - 7.3. Règles de pare-feu
 - 7.4. Systèmes de fichiers
 - 7.5. Périphériques de stockage
 - 7.6. Contenus de fichiers

- 7.7. Archives
- 7.8. Tâches planifiées
- 7.9. Sécurité
- 7.10. Utilisateurs et groupes

1. Mise en place de la topologie de lab

1.1. Topologie de lab



Topologie du lab Ansible

1.2. Solution de lab

Il est recommandé d'utiliser Libvirt/KVM selon le chapitre [Monter un lab Linux pour Ansible](#).

1.3. Mode ad-hoc

La commande `ansible` offre la possibilité d'exécuter des modules ansible de manière "ad-hoc", c'est à dire tâche par tâche sur un groupe d'hôtes. L'intérêt est de pouvoir exécuter la même tâche en parallèle sur un inventaire (constitué d'un certain nombre de cibles) en tout ou en partie.

En général, la commande s'écrit `ansible` suivie d'un hôte ou d'un groupe d'inventaire suivie `-m` `nom_de_module` suivie l'option d'argument `-a "clé=valeur clé=valeur clé=valeur"`, par exemple de manière formelle :

```
ansible <group_inventaire> -m nom_de_module -a "clé=valeur clé=valeur clé=valeur"
```

Concrètement :

```
ansible localhost -m debug -a "msg='Hello World'"
localhost | SUCCESS => {
  "msg": "Hello World"
```

```
}
```

La commande ansible attend comme argument un groupe cible de l'inventaire, une option de module et ses arguments.

On trouvera aussi des options utiles de la ligne de commande comme celles-ci :

- `-b`, `--become` active l'élévation de privilèges (`sudo`)
- `-e` qui permet de placer des variables ponctuelles ou un fichier de variables,
- `-o` qui offre la sortie standard en une seule ligne,
- ou encore `-l`, `--limit` qui permet de limiter la tâche à certains hôtes seulement de l'inventaire,
- ou encore `-v` ou `-vvv` pour la verbosité,
- `-i` précise le chemin du fichier d'inventaire.

Ansible génère des valeurs de retour à la suite de l'exécution des tâches et des modules.

1.4. Documentation des modules

Pour entamer cette activité, il est conseillé de prendre le temps de lire la documentation disponible sur votre machine de contrôle avant toute exécution avec la commande `ansible-doc <nom du module>`. On sera attentif aux arguments obligatoires (*mandatory*, `=`) et aux exemples proposés par rapport à l'objectif à atteindre.

2. Configuration du Contrôleur

La configuration du projet sur le contrôleur a toute son importance. Dans cette étape on propose de créer le projet avec Ansible lui-même, tant que faire se peut ... On comprendra peut-être mieux que la solution Ansible joue le rôle d'interface intelligible entre l'infrastructure et ses administrateurs. Cette étape n'est pas nécessaire si le contrôleur est déjà approvisionné.

Nous avons besoin de deux fichiers : un inventaire et un fichier de configuration que nous allons créer dans un dossier dédié.

2.1. Installation de Ansible sur controller

Sur "controller", nous allons travailler dans le dossier `~/lab`, c'est une manière d'initier notre projet Ansible :

```
mkdir ~/lab  
cd ~/lab
```

Il est nécessaire d'identifier la distribution CentOS 8 :

```
cat /etc/os-release ; echo $PRETTY_NAME
```

```
CentOS Linux 8 (Core)
```

Si le “controller” est bien une machine Centos 8, on peut passer à l’installation du dépôt “Extra” nommés *epel-release* :

```
dnf -y install epel-release
```

Enfin, Ansible s’installe avec le gestionnaire de paquets :

```
dnf -y install ansible
```

Vérifions la version du moment :

```
ansible --version | head -1  
ansible 2.9.15
```

2.2. Premières tâches ad-hoc

On aurait pu créer ce dossier `~/lab` avec le module `ansible.builtin.file` et son argument `state=directory` :

```
ansible -m file -a "path=~/lab state=directory" localhost
```

Avec Ansible, on aurait utilisé le module `ansible.builtin.setup` qui récupère les “facts” pour obtenir la version de la distribution :

```
ansible -m setup -a "filter=ansible_distribution*" localhost
```

2.3. Création d’un fichier d’inventaire

Avant toute chose, un projet de gestion avec ansible a besoin d'un inventaire pour gérer des hôtes distants. On notera que la cible "localhost" (127.0.0.1) est implicite, c'est-à-dire qu'elle ne doit pas nécessairement être citée dans l'inventaire pour en assurer la gestion.

Pour cette opération de création d'inventaire sur le contrôleur, on utilisera le module `ansible.builtin.copy` avec l'argument `content` qui crée le fichier en ajoutant du contenu. La séquence `\n` réalise des retours charriot dans le texte.

L'option `-e` ou `--extra-vars` dans le format `"user=${user}"` permet de passer le contenu des variables Bash dans des variables Jinja2 que les arguments des modules peuvent comprendre.

Ici, la cible est "localhost" et n'a pas besoin d'être défini dans l'inventaire.

```
user="root"
ansible -m ansible.builtin.copy -e "user=${user}" -a "dest=~/.lab/inventory content='[nodes]\nnode0\nnode1\nnode2\n\n[all:vars]\nansible_connection=ssh\nansible_user={{ user }}\nansible_ssh_pass=testtest\n'" localhost
```

Veillez vérifier vous-même :

```
cat ~/.lab/inventory
[nodes]
node0
node1
node2

[all:vars]
ansible_connection=ssh
ansible_user=root
ansible_ssh_pass=testtest
```

Cet inventaire désigne trois hôtes :

- `node0`
- `node1`
- `node2`

Il désigne explicitement aussi deux groupes :

- `nodes` qui comprend les hôtes `node0`, `node1` et `node2`.
- `all` avec des variables qui portent sur tous les hôtes de l'inventaire.

On sera attentif aux variables d'inventaire du groupe `all` :

- Le type de connexion en SSH : `ansible_connection=ssh`

- Le nom d'utilisateur pour se connecter `ansible_user=root`
- Le mot de passe de l'utilisateur de connexion : `ansible_ssh_pass=testtest`

2.4. Création d'un fichier de configuration minimal

De manière semblable avec le module `ansible.builtin.copy`, on peut créer un fichier de configuration par défaut `ansible.cfg`

```
ansible -m ansible.builtin.copy -a "dest=~/.lab/ansible.cfg content='[defaults]\ninventory = ./inventory\nhost_key_checking = False\n#private_key_file = ~/.nodes.key\ndeprecation_warnings=False\n'" localhost
```

Veuillez vérifier vous-même :

```
cat ~/.lab/ansible.cfg
[defaults]
inventory = ./inventory
host_key_checking = False
#private_key_file = ~/.nodes.key
deprecation_warnings=False
```

Ce fichier indique l'emplacement de l'inventaire (`./inventory`), désactive la vérification des hôtes SSH et désactive les avertissement concernant les changements futurs du comportement de Ansible. Tant que la clé publique de "controller" n'est pas placée sur le compte l'utilisateur de gestion "ansible" des deux noeuds, le paramètre de clé privée est désactivé dans le fichier de configuration. C'est celle de l'utilisateur root, déjà connue des cibles, qui est utilisée pour l'instant. Nous créerons la clé privée d'un utilisateur de gestion dédié dans les étapes suivantes.

2.5. Test de connectivité

Le module `ansible.builtin.ping` teste la connexion de gestion par Ansible vers la cible, avec une tentative de connexion qui vérifie l'existence de Python. Attention, le résultat dépend du plugin de connexion utilisé. On sera attentif aux paramètres de connexion utilisateur, de ses privilèges et de mode d'élévation de privilèges, de mot de passe ou de clé privée de la configuration, etc. La réponse attendue est "pong". Attention, il ne s'agit pas d'un test ICMP Echo Request !

En toute logique, si le lab est bien monté, un résultat positif "pong" doit vous parvenir :

```
ansible all -m ping
```

```
node0 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/libexec/platform-python"
  },
  "changed": false,
  "ping": "pong"
}
node2 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/libexec/platform-python"
  },
  "changed": false,
  "ping": "pong"
}
node1 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/libexec/platform-python"
  },
  "changed": false,
  "ping": "pong"
}
```

Notez que le binaire “ansible” demande un hôte ou un groupe d’inventaire, ici `all` :

```
ansible -m ping all
```

[Le code source du module Ping sur Github](#) nous indique que la charge utile du module tient en une vingtaine de lignes.

3. Commandes brutes

Toutes les actions qui suivent se déroulent désormais sur le “controller” dans le dossier de travail `~/lab` qui dispose d’un fichier de configuration et d’un inventaire.

```
cd ~/lab
```

3.1. Commandes brutes

Si on ne précise pas le nom du module dans la commande `ansible`, c'est le module `ansible.builtin.raw` par défaut qui est utilisé avec les commandes brutes comme option d'argument.

Le module `raw` exécute les arguments directement en “brut” (*low-down and dirty*) avec le plugin de connexion, ici dans une session SSH.

Quand on ne précise pas le module avec `-m <module>` et que l'option `-a` est utilisée, Ansible suppose alors le module `raw` :

```
ansible nodes -a "hostname"
```

```
ansible nodes -a "df -h"
```

Installer Python3-pip comme pré-requis à une gestion ansible :

```
ansible nodes -a "dnf -y install python3-pip"
```

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

```
ansible all -a "date"
```

```
ansible all -a "cat /etc/redhat-release"
```

```
ansible all -a "mount"
```

```
ansible all -a "getenforce"
```

3.2. Redémarrage en mode brut

Pour redémarrer tous les serveurs dans le groupe `[nodes]` :

```
ansible nodes -a "/sbin/reboot"
```

Il est logique que cette opération brutale de redémarrage rompe la connexion entre le contrôleur et ses nœuds en cours gestion et rende un retour “UNREACHABLE” ou “FAILED/rc=-1” !


```
nnode2 | FAILED | rc=-1 >>
Failed to connect to the host via ssh: ssh: connect to host 192.168.122.223 port 22: Connection reset by peer
node0 | FAILED | rc=-1 >>
Failed to connect to the host via ssh: ssh: connect to host 192.168.122.37 port 22: Connection refused
node1 | UNREACHABLE! => {
  "changed": false,
  "msg": "Failed to connect to the host via ssh: Shared connection to 192.168.122.222 closed.",
  "unreachable": true
}
```

Nous verrons dans la suite comment mieux gérer le redémarrage des systèmes sans générer une erreur de connexion.

3.3. Faire une pause

Le module `ansible.builtin.pause` permet de faire une pause dans le livre de jeu.

```
ansible nodes -a "/sbin/reboot"
ansible -m pause -a "minutes=2" nodes
```

3.4. Attendre une connection

Le module `ansible.builtin.wait_for_connection` est un module qui attend qu'un système distant soit joignable/utilisable.

```
ansible nodes -a "/sbin/reboot"
ansible nodes -m wait_for_connection
```

Plus fin, on peut attendre la bannière du serveur SSH avec le module `ansible.builtin.wait_for` qui attend une condition avant de continuer.

```
ansible nodes -a "/sbin/reboot"
ansible nodes -m wait_for -a "port=22 host='{{ inventory_hostname }}' search_regex=OpenSSH delay=60"
```

On remarquera ici l'usage de la variable magique `inventory_hostname` qui nous assure que le nom d'hôte référencé dans l'inventaire servira à la connexion.

3.5. Redémarrer un hôte avec le module approprié

Le module `ansible.builtin.reboot` redémarre de manière appropriée le système distant :

```
ansible nodes -m reboot
```

3.6. Processus simultanés

Par défaut, Ansible utilise 5 processus simultanés (forks).

```
ansible-config dump | grep FORKS
```

```
DEFAULT_FORKS(default) = 5
```

On peut manipuler ce paramètre comme variable d'environnement, dans le fichier de configuration ou directement sur la ligne de commande avec `-f` ou `--forks`. Ici pour redémarrer les serveurs du groupe `nodes` avec 10 forks :

```
ansible nodes -m reboot -f 10
```

Le nombre de communication simultanées peut être contrôlé dans le fichier de configuration dans la section `[default]` :

```
[defaults]
forks = 10
```

Il est également possible de contrôler la manière dont les tâches se succèdent sur les cibles avec les plugins de stratégie "serial" ou "free". Avec "serial", il est possible de diviser la charge simultanée de connexion à partir du contrôleur de manière relative ou absolue.[1](#)

3.7. Autres modules de commandes

On trouve au moins deux autres modules de commandes comme `ansible.builtin.command` et `ansible.builtin.shell`.

Le module `command` permet d'exécuter une commande ou un script dans le même shell que les autres tâches alors que le module `shell` ouvre un nouveau shell (`/bin/sh`). Le module `shell` autorise probablement des commandes plus complexes avec des tubes et des redirections par exemple sur une même ligne.

```
ansible localhost -m command -a "echo test"
```

```
ansible localhost -m shell -a "echo test | tee /tmp/test "
```

En comportement normal, s'ils réussissent (`rc=0`), quel que soit le véritable état, les modules `shell` et `command` offrent un état "CHANGED" :

```
ansible localhost -m command -a "echo test"
localhost | CHANGED | rc=0 >>
test
```

Même s'ils ne changent rien en cas de réussite (`rc=0`), ces modules donnent toujours un état "CHANGED", ce qui rend par nature la tâche non-idempotente.² En effet, ces modules ne peuvent pas être idempotents naturellement. Heureusement, les livres de jeu permettent de jouer avec ce comportement. On tentera d'éviter l'usage de ces modules en utilisant des modules natifs pour la tâche ou en développant soi-même des modules plus robustes.

4. Utilisateurs de gestion

4.1. Gérer les utilisateurs et les groupes

Les modules `ansible.builtin.user` et `ansible.builtin.group` gèrent les utilisateurs et les groupes.

Ici on crée un utilisateur "louise" dans le groupe "team" :

```
ansible nodes -m group -a "name=louise state=present"
ansible nodes -m group -a "name=team state=present"
ansible nodes -m user -a "name=louise groups=team,louise createhome=yes append=true"
```

Ajouter un utilisateur dans un groupe :

```
ansible nodes -m user -a "name=louise groups=wheel append=true"
```

Supprimer un utilisateur et son groupe :

```
ansible nodes -m user -a "name=louise state=absent remove=yes"
ansible nodes -m group -a "name=team state=absent"
```

4.2. Configurer de sudo

Au préalable, on s'assurera que sudo est installé avec le module [ansible.builtin.package](#) :

```
ansible nodes -m package -a "name=sudo"
```

Notre intention est de créer un utilisateur de gestion qui pourrait obtenir une élévation de privilège sans qu'un mot de passe soit demandé. On ne discutera pas ici de l'opportunité de cette configuration.

Tentons de trouver de manière récursive une entrée "ALL=(ALL) NOPASSWD:ALL" dans un fichier situé dans le chemin `/etc` avec le module Ansible [ansible.builtin.find](#) qui identifie le rôle sudo sans mot de passe :

```
ansible node0 -m find -a "paths=/etc contains='.*ALL=\\(ALL\\)\\s*NOPASSWD\\:\\s*ALL' recurse=yes"
```

On apprend que le fichier `/etc/sudoers` contient cette occurrence :

```
ansible node1 -a "cat /etc/sudoers"
```

On pourrait créer une entrée spécifique dans ce fichier ou encore adapter les droits du groupe `wheel` et ajouter l'utilisateur dans le groupe wheel.

Une autre solution consiste à créer un fichier dans `/etc/sudoers.d/`. La dernière recherche nous donne un autre résultat sur le fichier `/etc/sudoers.d/90-cloud-init-users`. Comment est-il écrit, d'où vient-il ?

```
ansible node1 -a "cat /etc/sudoers.d/90-cloud-init-users"
```

```
node1 | CHANGED | rc=0 >>
# Created by cloud-init v. 19.4 on Sat, 21 Nov 2020 11:36:16 +0000
```

```
# User rules for ansible
ansible ALL=(ALL) NOPASSWD:ALL
```

Créons plutôt un utilisateur “ansible” pour une gestion privilégiée :

```
ansible nodes -m package -a "name=sudo"
ansible nodes -m user -a "name=ansible"
ansible -m ansible.builtin.copy -a "dest=/etc/sudoers.d/ansible content='ansible ALL=(ALL) NOPASSWD:ALL'"
nodes
```

Le module `ansible.builtin.file` permet d’effacer le fichier `/etc/sudoers.d/90-cloud-init-users` :

```
ansible -m file -a "path=/etc/sudoers.d/90-cloud-init-users state=absent" nodes
```

4.3. Création des clés SSH de gestion

Controller doit disposer d’une clé privée associée à une clé publique placée dans le fichier `~/.ssh/authorized_keys` d’un utilisateur de gestion sur les cibles, ici `root` bien que cela ne soit pas recommandé.

La création d’une paire de clés SSH peut se réaliser avec le module `community.crypto.openssh_keypair` en utilisant les arguments `path` et `size`. Le module `ansible.builtin.file` permet de manipuler les propriétés des fichiers et des dossiers, comme déjà évoqué.

```
ansible -m openssh_keypair -a "path=~/.nodes.key size=2048" localhost
ansible -m file -a "path=~/.nodes.key mode=0600" localhost
```

4.4. Copie de la clé publique

Le module `ansible.posix.authorized_key` se charge de placer la clé publique du contrôleur dans `/home/ansible/.ssh/authorized_keys` sur chaque noeud, car c’est l’utilisateur `ansible` qui est utilisé pour la gestion :

```
ansible -m authorized_key -a "user=ansible key='{{ lookup('file', '~/.nodes.key.pub') }}'" nodes
```

On remarque l’usage d’une variable Jinja2 avec un filtre `lookup` qui lit un fichier local : la clé publique qui a été générée précédemment sur le système de fichier local.

4.5. Modification du fichier d'inventaire

Nous avons besoin de modifier le fichier d'inventaire pour exploiter nos changements. Il devrait ressembler à ceci :

```
[nodes]
node1
node2

[all:vars]
ansible_connection=ssh
ansible_user=ansible
ansible_become=yes
ansible_become_user=root
ansible_become_method=sudo
```

Dans cette opération, on utilisera le module [ansible.builtin.lineinfile](#) avec les arguments `regexp` et `state` pour effacer la ligne qui indique le mot de passe, ainsi que le module [community.general.ini_file](#) avec les paramètres `section`, `option` et `value` :

```
ansible localhost -m lineinfile -a "path=~/.lab/inventory regexp='^ansible_ssh_pass=.*$' state=absent"
ansible localhost -m lineinfile -a "path=~/.lab/inventory regexp='^ansible_user=' line='ansible_user=ansible'"
ansible localhost -m ini_file -a "path=~/.lab/inventory section='all:vars' option='ansible_become' value='yes'"
ansible localhost -m ini_file -a "path=~/.lab/inventory section='all:vars' option='ansible_become_user'
value='root'"
ansible localhost -m ini_file -a "path=~/.lab/inventory section='all:vars' option='ansible_become_method'
value='sudo'"
```

Veuillez vérifier vous-même :

```
cat ~/.lab/inventory

[nodes]
node1
node2

[all:vars]
ansible_connection=ssh
ansible_user=ansible
```

```
ansible_become=yes
ansible_become_user=root
ansible_become_method=sudo
```

4.6. Modification du fichier de configuration minimal

De manière semblable avec le module [ansible.builtin.replace](#), on peut modifier le fichier de configuration par défaut `ansible.cfg` en décommentant la ligne `private_key_file` :

```
ansible localhost -m replace -a "path=~/.lab/ansible.cfg regexp='#private_key_file.*$' replace='private_key_file=
~/nodes.key'"
```

Veuillez vérifier vous-même :

```
cat ~/.lab/ansible.cfg
[defaults]
inventory = ./inventory
host_key_checking = False
private_key_file = ~/.nodes.key
deprecation_warnings=False
```

Vérifions la connectivité :

```
ansible -m ping all
```

4.7. Paramètres d'élévation de privilèges

Par défaut, Ansible utilise le compte courant comme utilisateur distant. Notre inventaire utilise un autre compte (le compte distant "ansible").

On peut manipuler le type de connexion, le nom d'utilisateur, l'usage de l'élévation de privilège, le mot de passe de l'utilisateur et/ou la clé publique dans des variables d'inventaire ou de configuration comme nous l'avons configuré dans ce lab.

Si l'on désire contrôler l'élévation de privilèges sur la ligne de commande, on peut utiliser les options suivantes :

- `-u REMOTE_USER, --user REMOTE_USER`
- `-k, --ask-pas`
- `-b, --become`
- `-K, --ask-become-pass`
- `--become-user BECOME_USER`
- `--become-method BECOME_METHOD`

Pour se connecter avec l'utilisateur "ansible" :

```
ansible nodes -m reboot -u ansible
```

Redémarrer un serveur exigerait probablement une élévation de privilèges

```
ansible nodes -m reboot -u ansible --become
```

Si vous ajoutez `--ask-become-pass` ou `-K`, Ansible demande le mot de passe pour l'élévation de privilège (sudo/su/pfexec/doas/etc).

```
ansible nodes -m reboot -u ansible --become --ask-become-pass
```

5. Manipulation de fichiers

5.1. Trouver des fichiers

Nous avons déjà vu que le module [ansible.builtin.find](#) permet de trouver des fichiers en fonction d'un contenu identifié par expression rationnelle. Mais on l'utilise communément à la manière de la commande `find`.

Par exemple, pour trouver dans le dossier `/usr/share` de manière récursive tous les fichiers plus vieux de 4 semaines et plus grand ou égaux à 10 Mo.

```
ansible all -m find -a "paths=/usr/share age=4w size=10m recurse=yes"
```

5.2. Prendre des informations sur un fichier

C'est le module `ansible.builtin.stat` qui offre une sortie complète sur la nature d'un fichier ou d'un dossier :

```
ansible nodes -m stat -a "path=/etc/shadow"
```

```
ansible nodes -m stat -a "path=/etc"
```

```
ansible nodes -m stat -a "path=/opt/random"
```

5.3. Créer des dossiers et des fichiers

Sur `node0` uniquement nous créons un dossier `/tmp/test` :

```
ansible nodes -m file -a "dest=/tmp/test mode=644 state=directory"
ansible nodes -m stat -a "path=/tmp/test"
```

Et puis, nous créons un lien symbolique `/opt/tmp` qui pointe sur `/tmp/test` :

```
ansible nodes -m file -a "src=/tmp/test dest=/opt/tmp owner=root group=root state=link"
ansible nodes -m stat -a "path=/opt/tmp"
```

5.4. Fixer des droits

Toujours sur `node0` uniquement, nous créons trois utilisateurs "alpha", "beta" et "gamma" dans un groupe "omega", nous ajoutons notre clé publique dans chaque profil :

```
group="omega"
users="alpha beta gamma"
ansible node0 -e "group=${group}" -m group -a "name={{ group }}"
for user in ${users} ; do
  ansible node0 -e "user=${user}" -m group -a "name={{ user }}"
  ansible node0 -e "group=${group} user=${user}" -m user -a "name={{ user }} group={{ user }} groups={{
group }} append=yes"
  ansible node0 -e "user=${user}" -m authorized_key -a "user={{ user }} key='{{ lookup('file',
'~/nodes.key.pub') }}'"
done
```

Toujours sur `node0` uniquement, nous retirerons le lien symbolique `/opt/tmp` vers `/tmp/test` :

```
ansible node0 -m file -a "src=/tmp/test dest=/opt/tmp state=absent"
```

Et nous configurons le dossier partagé avec le Sticky bit et le SGID activés ; aussi, nous créons un lien symbolique `/opt/omega` vers le dossier partagé.

```
ansible node0 -m file -a "dest=/tmp/test state=directory group=omega mode=3770"
ansible node0 -m file -a "src=/tmp/test dest=/opt/omega group=omega state=link follow=yes"
ansible node0 -m stat -a "path=/opt/omega"
```

On peut réaliser quelques tests avec le module `shell` :

```
ansible all -i 'node0,' -m shell -u alpha -a "echo alpha > /opt/omega/init.txt"
ansible all -i 'node0,' -m shell -u beta -a "echo beta >> /opt/omega/init.txt"
ansible all -i 'node0,' -m shell -u gamma -a "cat /opt/omega/init.txt"
ansible all -i 'node0,' -m shell -u gamma -a "rm -f /opt/omega/init.txt"
ansible all -i 'node0,' -m shell -u beta -a "rm -f /opt/omega/init.txt"
ansible all -i 'node0,' -m shell -u alpha -a "rm -f /opt/omega/init.txt"
```

5.5. ACLs sur les fichiers

[ansible.posix.acl](#)

...

5.6. Effacer des dossiers et des fichiers

Un état “absent” efface le fichier ou le dossier de manière récursive par défaut :

```
ansible node0 -m file -a "dest=/opt/omega state=absent"
```

```
ansible node0 -m stat -a "path=/opt/omega"
```

5.7 Archiver des fichiers

```
for format in gz zip xz bz2 ; do
ansible nodes -e "format=${format}" -m archive -a "path=/usr/share/doc dest=/tmp/doc.{{ format }}"
format={{ format }}"
done
```

```
for format in gz zip xz bz2 ; do
ansible node0 -e "format=${format}" -m shell -a "ls -lh /tmp/doc."
done
```

5.8. Copier un fichier local sur l'hôte distant

```
ansible nodes -m ansible.builtin.copy -a "src=/etc/hostname dest=/tmp/hostname"
ansible nodes -m stat -a "path=/tmp/hostname"
```

5.9. Rapatrier un fichier sur le contrôleur

```
ansible node1 -m fetch -a "src=/etc/hostname dest=/tmp/{{ inventory_hostname }}"
ansible localhost -m stat -a "path=/tmp/node1"
```

5.10. Décompacter des archives

[ansible.builtin.unarchive](#) copying it from the local machine.

...

Synchroniser des fichiers

[ansible.posix.synchronize](#)

...

6. Manipuler des volumes

- [ansible.posix.mount](#) - Control active and configured mount points
- [community.general.filesystem](#) - Makes a filesystem
- [community.general.lvg](#) - Configure LVM volume groups
- [community.general.lvol](#) - Configure LVM logical volumes

6.1. Créer un disque virtuel léger en format fichier

- Taille du disque : 8 Go

```
ansible nodes -m shell -a "dd if=/dev/zero of=/opt/fs_1.img bs=1M seek=8192 count=0"
ansible nodes -m shell -a "losetup -fP /opt/fs_1.img"
```

6.2. Vérifier que les outils LVM2 sont installés

```
ansible nodes -m package -a "name=lvm2"
```

6.3. Créer un volume physique et un volume group

- nom du physical volume : `/dev/loop0`
- nom du volume group : `vg.fs_1`

```
ansible nodes -m lvg -a "vg=vg.fs_1 pvs=/dev/loop0 pesize=256K"
```

6.4. Ajout d'un volume logique

- taille du volume logique : 1 Go

- nom du volume logique : `lv.ext4`
- nom du volume group : `vg.fs_1`

```
ansible nodes -m lvol -a "vg=vg.fs_1 lv=lv.ext4 size=1g"
```

6.5. Système de fichiers EXT4

- système de fichier : `ext4`

```
ansible nodes -m filesystem -a "dev=/dev/vg.fs_1/lv.ext4 fstype=ext4 resizefs=yes force=yes"
```

6.6. Point de montage et fstab

- périphérique : `/dev/vg.fs_1/lv.ext4`
- système de fichier : `ext4`
- point de montage : `/mnt/fs_1/ext4`

```
ansible nodes -m file -a "path=/mnt/fs_1/ext4 state=directory"
```

```
ansible nodes -m mount -a "path=/mnt/fs_1/ext4 src=/dev/vg.fs_1/lv.ext4 fstype=ext4 opts=defaults  
state=mounted boot=yes"
```

```
ansible node1 -m setup -a "filter=ansible_mounts"
```

6.7. Même exercice avec un volume logique XFS

- taille du volume logique : `1 Go`
- nom du volume logique : `lv.xfs`
- nom du volume group : `vg.fs_1`
- périphérique : `/dev/vg.fs_1/lv.xfs`
- système de fichier : `xfs`
- point de montage : `/mnt/fs_1/xfs`

```
ansible nodes -m lvol -a "vg=vg.fs_1 lv=lv.xfs size=1g"
```

```
ansible nodes -m filesystem -a "dev=/dev/vg.fs_1/lv.xfs fstype=xfs resizefs=yes force=yes"
```

```
ansible nodes -m file -a "path=/mnt/fs_1/xfs state=directory"
```

```
ansible nodes -m mount -a "path=/mnt/fs_1/xfs src=/dev/vg.fs_1/lv.xfs fstype=xfs opts=defaults state=mounted  
boot=yes"
```

```
ansible node1 -m setup -a "filter=ansible_mounts"
```

6.8. Extension d'un volume logique

- taille du volume logique : **2 Go**
- nom du volume logique : `lv.xfs`
- nom du volume group : `vg.fs_1`
- périphérique : `/dev/vg.fs_1/lv.xfs`
- système de fichier : `xfs`
- point de montage : `/mnt/fs_1/xfs`

```
ansible nodes -m lvol -a "vg=vg.fs_1 lv=lv.xfs size=2g"
```

```
ansible nodes -m filesystem -a "dev=/dev/vg.fs_1/lv.xfs fstype=xfs resizefs=yes force=yes"
```

```
ansible nodes -m file -a "path=/mnt/fs_1/xfs state=directory"
```

```
ansible nodes -m mount -a "path=/mnt/fs_1/xfs src=/dev/vg.fs_1/lv.xfs fstype=xfs opts=defaults state=mounted  
boot=yes"
```

```
ansible node1 -m setup -a "filter=ansible_mounts"
```

6.9. Eliminer la configuration

```
fsname="fs_1"
```

```
for fstype in ext4 xfs ; do
```

```
ansible nodes -e "fstype=${fstype} fsname=${fsname}" -m mount -a "state=absent path=/mnt/{{ fsname  
}}/{{ fstype }}" src=/dev/vg.fs_1/lv.{{ fstype }}"
```

```
ansible nodes -e "fstype=${fstype} fsname=${fsname}" -m file -a "path=/mnt/{{ fsname }}/{{ fstype }}  
state=absent"
```

```
ansible nodes -e "fstype=${fstype} fsname=${fsname}" -m lvol -a "vg=vg.{{ fsname }} lv=lv.{{ fstype }}  
state=absent force=yes"
```

```
done
```

```
ansible nodes -e "fstype=${fstype} fsname=${fsname}" -m lvg -a "vg=vg.{{ fsname }} pvs=/dev/loop0  
state=absent"
```

```
ansible nodes -e "fstype=${fstype} fsname=${fsname}" -m shell -a "losetup -D"
ansible nodes -e "fstype=${fstype} fsname=${fsname}" -m file -a "path=/opt/{ { fsname } }.img state=absent"
```

6.10. Créer un partage NFS

...

7. Récupérer des Facts

7.1. Facts sur les hôtes

Par défaut, le module `ansible.builtin.gather_facts` est exécuté implicitement dans les livres de jeu avec `ansible-playbook`. Ce comportement peut se contrôler dans un jeu ou dans la configuration.

Mais avec le binaire `ansible`, c'est le module `ansible.builtin.setup` qui affiche les variables collectées.

```
ansible node1 -m setup
```

L'argument `filter` est utile sur la ligne de commande dans un usage Ad Hoc.

```
ansible node1 -m setup -a 'filter=ansible_mounts'
```

```
ansible node1 -m setup -a "filter=*ipv4*"
```

```
ansible node1 -m setup -a "filter=ansible_distribution*"
```

```
ansible node1 -m setup -a "filter=ansible_swapfree_mb"
```

7.2. Module Debug

Le module `ansible.builtin.debug` permet d'afficher des messages. Il prend comme argument soit `var` qui affiche les variables ou `msg` qui affiche un message avec des variables.

Voyez la différence :

```
ansible nodes -m debug -a "var='inventory_hostname'"
```

```
ansible nodes -m debug -a "msg='Hello {{ inventory_hostname }}'"
```

8. Installer et gérer un service

8.1. Installation du service Chrony

Nous allons ici installer le service de temps Chrony sur Centos8 avec les modules [ansible.builtin.dnf](#) ou [ansible.builtin.yum](#) ou encore le module générique [ansible.builtin.package](#) :

```
ansible all -m dnf -a "name=chrony state=present"
```

```
ansible all -m package -a "name=chrony state=present"
```

8.2. Démarrer et activer le service

Démarrer et activer le service avec les modules [ansible.builtin.service](#) ou [ansible.builtin.systemd](#) :

```
ansible all -m service -a "name=chronyd state=started enabled=yes"
```

```
ansible all -m systemd -a "name=chronyd state=started enabled=yes"
```

Pour synchroniser les hôtes, on fera appel aux modules [service](#) et [raw](#) :

```
ansible all -m service -a "name=chronyd state=stopped"
ansible all -a "chronyd -q 'server 0.fr.pool.ntp.org iburst'"
ansible all -m service -a "name=chronyd state=started"
```

Ou encore :

```
ansible all -a "chronyc -a makestep"
```


Si nous étions invité à réaliser ces opérations sur des cibles Debian/Ubuntu ou en Centos 7, on proposerait ceci :

Installation de NTPD sur Debian/Ubuntu :

```
ansible all -m apt -a "name=openntpd state=present"
ansible all -m service -a "name=openntpd state=started enabled=yes"
```

Installation de NTPD sur RHEL7 :

```
ansible all -m yum -a "name=ntp state=present"
ansible all -m service -a "name=ntpd state=started enabled=yes"
```

Et puis les tentatives de synchronisation :

```
ansible all -m service -a "name=ntpd state=stopped"
ansible all -a "ntpdate -q 0.eu.pool.ntp.org"
ansible all -m service -a "name=ntpd state=started"
```

9. Limites et tâches en arrière-plan

9.1. Limiter une tâche à un hôte

Dans cette démarche on exécute l'inventaire contre un groupe d'hôte (ici "all") en le limitant uniquement à "node1" :

```
ansible all -m service -a "name=chronyd state=restarted" --limit node1
```

L'option `--limit` accepte des expressions rationnelles (regex) :

```
ansible all -m service -a "name=chronyd state=restarted" --limit "node*"
```

L'option `--limit` accepte une liste :

```
ansible all -m service -a "name=chronyd state=restarted" --limit "node1,node2"
```

9.2. Tâches en arrière-plan

L'option `-B` permet d'activer la tâche en arrière-plan avec un délai maximum.

```
ansible nodes -B 3600 -m dnf -a "name=*" state=latest"
```

10. Tâches planifiées

- [ansible.builtin.cron](#) – Manage cron.d and crontab entries
- [ansible.posix.at](#) – Schedule the execution of a command or script file via the at command
- [community.general.cronvar](#) – Manage variables in crontabs

...

11. Manipulation de contenu de fichiers

- [ansible.builtin.assemble](#) – Assemble configuration files from fragments
- [ansible.builtin.blockinfile](#) – Insert/update/remove a text block surrounded by marker lines
- [ansible.builtin.lineinfile](#) – Manage lines in text files
- [ansible.builtin.replace](#) – Replace all instances of a particular string in a file using a back-referenced regular expression
- [ansible.builtin.copy](#) – Copy files to remote locations
- [ansible.builtin.tempfile](#) – Creates temporary files and directories
- [ansible.builtin.template](#) – Template a file out to a remote server

...

12. Installation d'un service applicatif LAMP

- [ansible.builtin.iptables](#) – Modify iptables rules
- [ansible.posix.firewalld](#) – Manage arbitrary ports/services with firewalld

- [community.general.iptables_state](#) – Save iptables state into a file or restore it from a file

...

- [ansible.posix.seboolean](#) – Toggles SELinux booleans
- [ansible.posix.selinux](#) – Change policy and state of SELinux
- [community.general.sefcontext](#) – Manages SELinux file context mapping definitions
- [community.general.selinux_permissive](#) – Change permissive domain in SELinux policy
- [community.general.sellogin](#) – Manages linux user to SELinux user mapping
- [community.general.seport](#) – Manages SELinux network port type definitions

...

12.1. Installation du service Apache HTTPd

Sous Centos :

```
ansible nodes -m yum -a "name=httpd state=present"
```

12.2. Lancer le service Apache HTTPd

Sous Centos :

```
ansible nodes -m service -a "name=httpd state=started"
```

12.3. Tester le service Apache HTTPd

```
ansible nodes -m uri -a "url=http://{ansible_host} return_content=yes status_code=200,403"
```

12.4. Arrêter le service

Sous Centos :

```
ansible nodes -m service -a "name=httpd state=stopped"
```

12.5. Retirer Apache

Sous Centos :

```
ansible nodes -m yum -a "name=httpd state=absent"
```

12.6. Installation et gestion du service MariaDB

```
ansible db -m yum -a "name=mariadb-server state=present"
ansible db -m service -a "name=mariadb state=started enabled=yes"
```

12.7. Configurer les serveurs d'application

!!!

```
ansible nodes -m dnf -a "name=python2-mysql state=present"
ansible nodes -m dnf -a "name=python2-setuptools state=present"
ansible nodes -m easy_install -a "name=virtualenv executable=easy_install-2"
ansible nodes -m easy_install -a "name=django state=present virtualenv=/opt"
```

12.8. Pare-feu Firewallld

...

12.9. Pare-feu Iptables

!!!

```
ansible db -a "iptables -F"
ansible db -a "iptables -A INPUT -s 192.168.60.0/24 -p tcp -m tcp --dport 3306 -j ACCEPT"
```

Comment traduire les commandes iptables en module Ansible ?

```
ansible db -m iptables -a "flush=yes"
```

```
ansible db -m iptables -a "chain=INPUT protocol=tcp source=192.168.60.0/24 destination_port=3306  
jump=ACCEPT"
```

```
ansible db -m yum -a "name=MySQL-python state=present"
```

```
ansible db -m mysql_user -a "name=django host=% password=12345 priv=*.*:ALL state=present"
```

1. [Setting the batch size with serial ↩](#)
2. “Être idempotent permet à une tâche définie d’être exécutée une seule fois ou des centaines de fois sans créer un effet contraire sur le système cible, ne provoquant un changement qu’à une seule reprise. En d’autres mots, si un changement est nécessaire pour obtenir le système dans un état désiré, alors le changement est réalisé ; par contre si le périphérique est déjà dans l’état désiré, aucun changement n’intervient. Ce comportement est différent des pratiques de scripts personnalisés et de copier/coller de lignes de commandes. Quand on exécute les mêmes commandes ou scripts sur un même système de manière répétée, le taux d’erreur est souvent élevé.” Extrait de: Jason Edelman, “Network Automation with Ansible”, O’Reilly Media, 2016. [↩](#)

Revision #1

Created 3 October 2021 21:08:54 by garfi

Updated 3 October 2021 21:10:14 by garfi