



Ansible



Introduction

Créé en 2012 (2015 repris par Redhat) par Michael DeHaan (Cobler, outil de provisionnement)

Ansible = Infrastructure as code + déploiement de configurations + installations

A base de python et de SSH

Documentation : <https://docs.ansible.com/>

orchestrateur basé sur du push > pas d'agent = serveur distant pousse les informations

à la différence des outils à base d'agents > pull (puppet etc..)

concurrents : puppet, chef, capistrano, saltstack



Introduction

simplicité lié à l'utilisation de SSH

intégration facile dans les outils de CI/CD

facilité d'utilisation à base de fichiers yaml

de très nombreux modules et une très forte communauté (notamment via ansible galaxy)

différentes notions et définitions : inventory + playbook + rôles

inventory > playbook < rôles

des outils : ansible vault, ansible playbook, ansible galaxy, ansible doc



Introduction

installation : via les sources, les paquets ou la librairie python (pip)

système de templating = jinja3 (python), équivalent à erb pour puppet (ruby)

modules pour de nombreux outils :

- postgres
- vmware
- aws
- network
- grafana
- mysql...

également utilisable pour récupérer des données sur les serveurs

Idempotence et Stateless



Idempotence : quand je lance deux fois le même playbook, actions la première fois et la seconde ne change rien si la situation est restée identique lors du premier lancement (affiche OK dans TASK)

Affiché dans le ok en PLAY RECAP

Donc si on rejoue un playbook, vérifier qu'il y a bien changed=0 et que des ok (avec skipped possible)

Tout doit être décrit que cela existe ou non

Stateful : tout ce qui n'est pas décrit et donc j'ai connaissance qu'il ait existé ne doit plus exister

Ansible est **Stateless** : ne stocke pas son état \neq Terraform stateful

Idempotence et Stateless



Exemple :

Je désire créer une VM → Ansible la crée et c'est tout, il ne va pas créer un état de ce qu'il a créée (par exemple dans une database)

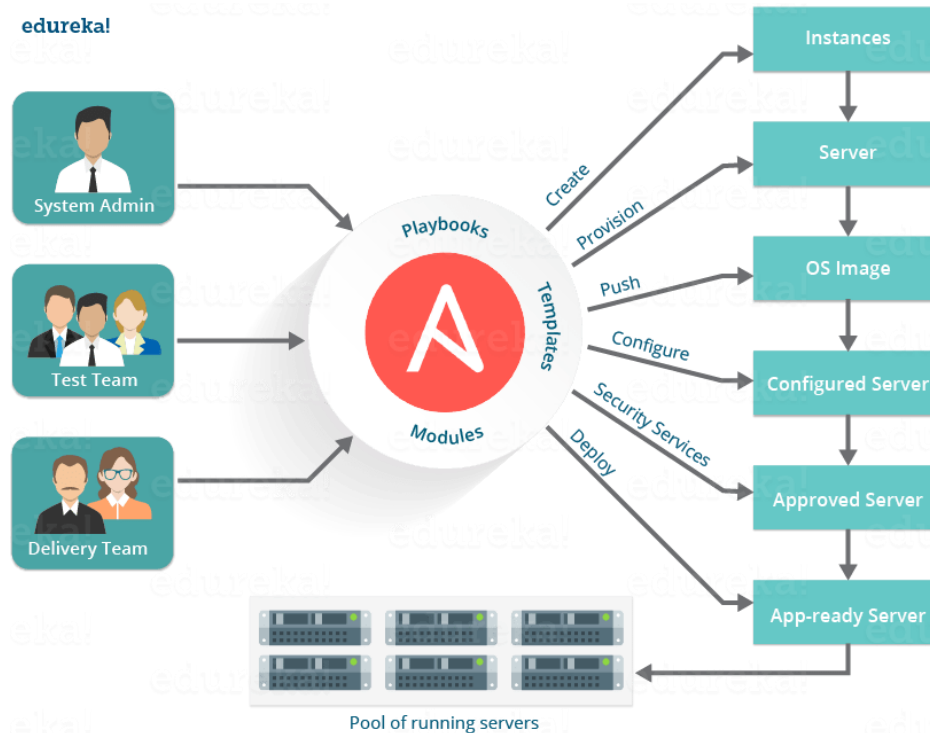
Je supprime la variable de la VM → Ansible ne reconnaît pas le système qu'il a mis en place (ne se réfère pas à une database comme Terraform par ex pour savoir ce qui existe ou non). Sauf si vous mettez en place un système de référencement pour connaître le parc... Mais risqué si des éléments n'avaient pas été créés par Ansible

Donc ce concept est le premier indicateur de choix



Notions et Définitions

Notions et Définitions



Notions et Définitions



Control node :

- * noeud disposant de ansible et permettant de déployer
- * accès ssh aux autres machines (bastions...)
- * password ou clef ssh
- * sécurité importante

Managed nodes :

- * serveurs cibles
- * permet la connexion ssh
- * élévation de privilèges via le user

Inventory :

- * inventaires des machines (ip, dns)
- * format ini (plat) ou format yaml
- * et les variables (host_vars et group_vars)
- * statique (fichiers) ou dynamique (api via script)
- * utilisation de patterns possible (srv-pg93-0[0-2])

Notions et Définitions



Groupes :

databases...)

- * dans un inventaire les machines peuvent être regroupées (serveur web, databases...)
- * possibilité de créer différents niveaux > arbre (parents/enfants)
- * groupe racine = all

Group Vars :

.yaml)

- * variables d'un même groupe
- * définie dans le fichier central d'inventory (pas obligatoire d'ajouter l'extension .yaml)
- * ou dans un répertoire spécifique (reconnu par ansible)

Host Vars :

groupe

- * variables spécifiques à un serveur en particulier
- * plus spécifique en terme de périmètre que le group_vars
- * surcharge d'autres variables définies plus haut dans l'arbre - ex -

Notions et Définitions



exemple d'inventory :

```

inventory.yml (description des machines, on peut changer le nom de ce fichier)

host\_vars/

group\_vars/

```

Garder les noms de fichiers `host_vars` et `group_vars`, car le fichier d'inventaire sera capable d'utiliser ces fichiers à côté

Notions et Définitions



Tasks :

- * Faire jouer ces éléments dans l'inventory
- * task = action la plus fine possible
- * actions variées (créer user, group, passer des commandes, utiliser des modules, copier fichier...)
- * format yaml

Tout est en yaml sur Ansible sauf sur l'inventory où .ini est possible

Notions et Définitions



Modules :

- * ensemble d'actions ciblées sur une utilisation commune (une action appelle un module comme générer un user, ansible reconnaît le module avec le nom)
- * pour un outil donnée : ex. postgres, mysql, vmware... (ex:plusieurs modules pour postgres)
- * chacune de ses actions est utilisable via une task
- * chaque action prend des options
- * les actions peuvent fournir un retour (id, résultat...)
- * fournis par ansible pour l'essentiel
- * peuvent être chargés spécifiquement
- * contribution possible auprès des mainteneurs

Notions et Définitions



Rôles :

- * ensemble d'actions coordonnées pour réaliser un ensemble d'actions cohérentes (installer nginx et le configurer etc)
- * organisé en différents outils (tasks, templates, handlers, variables (default ou non), meta)
- * peuvent être partagés sur le hub ansible galaxy (plus partagés = plus universels)
- * il vaut mieux les versionner
- * l'idéal est de les segmenter : pour être autonomes, pour les jouer indépendamment les uns des autres (ex: créer un user postgres pour plusieurs bases différentes)
- * Possible de faire un rôle avec une seule tâche à l'intérieur
- * Pour Git, utiliser un dépôt par rôle, par playbook...

Notions et Définitions



Playbooks :

- * un fichier (et rien d'autre...)
- * applique des rôles à un inventory (des variables à une liste de machines)
- * partie cruciale inventory > playbook < rôles : il peut faire ainsi jouer plusieurs rôles à un groupe de machines par exemple
 - * peut contenir des variables (à éviter)
 - * peut contenir des tasks (à éviter)
 - * peut contenir des conditions (à éviter)
 - * Essayer d'avoir un playbook le plus épuré possible, juste avoir la correspondance entre groupes et rôles

Plugins :

- * modifie ou augmente les capacités de ansible
- * utilisés plus rarement
- * de différentes manières : output, inventory dynamique, stratégie de déploiement , tests..

Notions et Définitions



IMPORTANT de comprendre clairement les différents termes car cela mène directement aux **bonnes pratiques**

Facilite l'intégration de personnes dans les équipes, l'intégration de rôles...

Partager des fichiers Ansible

Récupérer des éléments du Hub Galaxy



Installation d'Ansible



Controller node :

- * Python ≥ 2.7
- * tout sauf windows
- * ssh/scp (ou sftp)

Managed node : Python ≥ 2.6 (on peut ne pas avoir python au premier run)

Documentation : https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html

Différents types d'installations :

- * paquets des distributions (dans la plupart des distrib)
- * librairie python (pip)
- * binaire (pas recommandé)
- * éventuellement par docker (abandonné depuis 2 ans : pas très pratique, volumes à monter, pas optimal pour la sécurité...)

Les différentes releases : <https://releases.ansible.com/ansible/>



Installation d'Ansible

```
sudo apt install python3-pip  
pip3 install ansible
```

OU

```
sudo apt install ansible
```





Clefs SSH



Installation d'Ansible

1. Création de nos clés ssh
2. Lancement d'un Vagrantfile pour mettre en place nos nodes
3. Connection ssh aux nodes
4. Installation de pip sur le node 1 (manager)
5. Installation d'ansible sur le node 1 (manager)
6. Connection ssh entre node1 et node2 (slave)

1 . Création des clés ssh sur la machine hôte

```
sudo apt-get install openssh-server
```

```
sudo systemctl status sshd
```



Installation d'Ansible

1 . Création des clés ssh sur la machine hôte

```
sudo apt install openssh-server openssh-client
```

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

2 . Lancement d'un Vagrantfile pour mettre en place nos nodes

```
sudo apt install vagrant
```

```
vagrant --version
```



Installation d'Ansible



2 . Lancement d'un Vagrantfile pour mettre en place nos nodes

```
mkdir Ansible
```

```
cd Ansible
```

```
touch Vagrantfile
```

```
sudo vim Vagrantfile
```

```
vagrant up
```



Installation d'Ansible



3. Connection ssh entre ans1 et ans2 : depuis ans1

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

```
ssh-copy-id vagrant@192.168.15.10
```

```
ssh vagrant@IP
```

```
touch ~/.ssh/config  
chmod 600 ~/.ssh/config  
sudo vim ~/.ssh/config
```

```
Host ans2  
    User vagrant
```

Installation d'Ansible



4. Installation de pip sur ans1 (manager)

```
sudo apt install python3-pip
```

5. Installation d'ansible sur ans1 (manager)

```
pip3 install ansible
```

Se déconnecter d'ans1 et se reconnecter pour recharger tous les paquets nécessaires

```
ansible --version
```


Installation d'Ansible



remarque python interpreter - par défaut /usr/bin/python (or maintenant Os avec que du 3)

Dans inventory.ini :

```
ansible_python_interpreter=/usr/bin/python3
```

Si la machine distante ne dispose pas de python, on peut l'installer à distance :

```
ansible myhost --become -m raw -a "yum install -y python2(ou3)"
```

raw est un des seuls modules d'ansible qui n'utilise pas python

Doc : https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html#ansible-python-interpreter

Installation d'Ansible



6. Connection ssh entre node1 et node2

Pour éviter de retaper tout le temps la passphrase : agent ssh

```
eval `ssh-agent`
```

```
ssh-add -l
```

```
ssh-add
```

```
ssh-add -l
```

Installation d'Ansible



Exercice : Notez l'adresse IP des nodes ainsi que le nom de la machine (utilisez les commandes `hostname` et `ip address`) et mettre à jour le fichier `/etc/hosts`

Installation d'Ansible



On peut maintenant tester une commande ansible :

```
ansible -i "node2," all -m ping
```

Fichiers de configuration d'Ansible



Configuration de différentes manières :

- * ansible.cfg
- * cli

Notions de précedence :

et à différents endroits pour ansible.cfg (ordre de prise en compte)

- * éventuellement en définissant ANSIBLE_CONFIG
- * à l'endroit de votre playbook (dans dossier courant) ansible.cfg
- * ~/.ansible/ansible.cfg (dans le home)
- * /etc/ansible/ansible.cfg

Lorsque Ansible découvre un premier les autres sont ignorés

Fichiers de configuration d'Ansible : ansible.cfg



```
vim /etc/ansible/ansible.cfg
```

Il existe également un binaire ansible-config qui nous aide à paramétrer :

```
ansible-config --help
```

Fichiers de configuration d'Ansible : ansible.cfg



```
ansible-config view # voir quel ansible.cfg est pris en compte
ansible-config list # toutes les variables et leurs valeurs (ini est le fichier de
conf)
https://docs.ansible.com/ansible/latest/reference\_appendices/config.html

ansible-config dump # liste toutes les variables ansible
ansible-config dump --only-changed #valeurs par défaut modifiée
```

Pour voir les différentes sections de la configuration :

```
grep "\[" /etc/ansible/ansible.cfg
```

Fichiers de configuration d'Ansible : tuning



Sur des grosses infrastructures, fork = parallélisation

```
[defaults]  
forks = 30
```

Donc aussi attention 30 fois plus de risques

gather facts avec précaution : par défaut pour jouer un rôle Ansible va récupérer des informations du server pour setter des variables (l'os, sa version, les interfaces réseaux, les ip, bref presque 200). Il va les récupérer avant de lancer son script sur le server

```
gather_facts: no
```

Par contre, attention, en ne récupérant rien, on ne pourra pas utiliser les variables récupérées normalement avec un on sur les gather facts, si on a besoin de time, de l'os... Possibilité de les limiter

Fichiers de configuration d'Ansible : tuning



Si les gather facts n'évoluent pas souvent :

```
fact_caching = jsonfile
fact_caching_timeout = 3600 "exemple ici pendant une heure ne va pas évoluer"
fact_caching_connection = /tmp/mycachedir "information par serveur"
```

Donc le time ou les interfaces ajoutées par exemple ne seront pas à jour pendant une heure

Si désir de performance : **gather facts caching par redis**

```
fact_caching = redis
fact_caching_timeout = 3600
fact_caching_connection = host:port:db:password
```

Fichiers de configuration d'Ansible : tuning



Mitogen : permet de tuner au maximum le fork, le pipelining, paralléliser les sessions ssh

Doc : https://mitogen.networkgenomics.com/ansible_detailed.html

sur des très grosses infras, on utilise pull à la place du push, on installe an

exemple : vous avez une configuration sur votre pc linux et vous voulez ansible sur tous les serveurs et ansible va jouer en localhost. la reproduire sur votre dernier pc acheté. Vous la mettez sur un dépôt git, vous installez ansible sur la nouvelle machine et ensuite ansible récupère tout ça sur le dépôt et lance l'install complète.

- * cas ultime > ansible localhost >> ansible-pull (commande)
- * exécution en localhost
- * problème récupération des informations

CLI Ansible



peu utilisé (en proportion) au profit de ansible-playbook

permettre du test (ping, inventaire)

permet de jouer des tâches sommaires en appelant des modules (copie de fichier, installations...)

beaucoup d'options similaires à la commande ansible-playbook

```
ansible -i "node2," all -u vagrant -m ping
```

```
ping node
```

CLI Ansible



Principales options à connaître :

- * `-u` : **user** distant utilisé
- * `-b` : passer les commandes après en élévation de privilèges (sudo)
- * `-k` ou `--ask-pass` >connect via password user en SSH
- * `-K` ou `--ask-become-pass` > password pour élévation privilèges
- * `-C` ou `--check` : faire un dry **run** (**les commandes après ne sont pas créées**)
- * `-D` ou `--diff` : avoir un output de la diff (comme un git diff)
- * `--key-file` : lien direct vers fichier de la clef privée
- * `-e` ou `--extra-vars` : définir des variables directement
- * `--ask-vault-pass` : déchiffrer un **secret** vault (mot de passe pour déchiffrer le password)
- * `--vault-password-file` : fichier pour déchiffrer
- * `-f x` ou `--forks` : paralléliser (par défaut 5)
- * `-vvv` : verbose

CLI Ansible



```
ansible -i "node2," all -u vagrant -C -D -m  
ping
```

Avec cette commande, l'option -C permet de ne pas exécuter les commandes et -D affiche les différences si la commande avait été exécutée

Affichage sur une seule ligne : très utile pour récupérer des logs

```
ansible -i "node2," all -u vagrant -m ping --one-line
```

Passage de modules :

```
ansible -i "node2," all -u vagrant -m command -a uptime
```

CLI Ansible



Passage de variables :

```
ansible -i "node2," all -e "var1=bonjour" -m debug -a 'msg={{ var1 }}'
```

CLI Ansible



Passage de modules :

https://docs.ansible.com/ansible/2.9/modules/command_module.html#command-module

```
ansible -i "node2," all -u vagrant -m command -a pwd (ou ps...)
```

```
ansible -i "node2," all -u vagrant -m shell -a "ps aux | grep <username> | wc -l" -  
-one-line
```

```
ansible -i "node2," all -u vagrant -b -K -m raw -a "apt install -y mlocate"
```

CLI Ansible



Passage de modules :

```
ansible -i "ans2," all -b -m apt -a 'name=nginx state=latest'
```

vérifier avec 192.168.15.11 en local ou systemctl sur ans2

```
ansible -i "ans2," all -b -m service -a 'name=nginx state=stopped'
```


CLI Ansible



Passage de modules :

```
ansible -i "node2," all -m copy -a 'src=toto.txt dest=/tmp/titi.txt'
```

```
ansible -i "node2," all -m fetch -a 'src=/tmp/titi.txt dest=titi.txt flat=yes'
```

Et avec les gather_facts :

```
ansible -i "node2," all -m  
setup
```

```
ansible -i "node2," all -m setup -a "filter=ansible_distribution*"
```



CLI Ansible : Pratique

CLI Ansible : Pratique



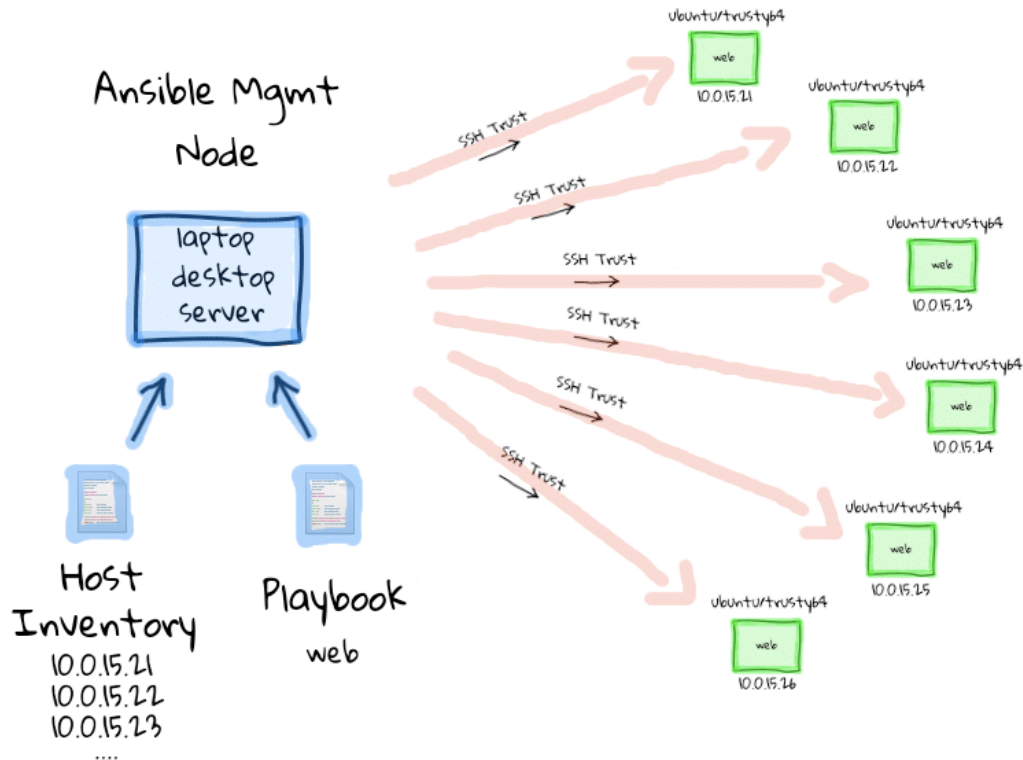
1. Vérifier la liste de vos nodes dans le répertoire d'ansible
2. Exécuter la commande **free -h du module command avec son option a** sur tous les nodes
3. Exécuter la commande sur tous les nodes : **date, sleep 5 ; date** avec le module **shell**
4. Copier le fichier /etc/hosts de votre serveur Ansible vers un fichier /tmp/MACHINES à l'aide du module **copy sur tous les nodes**
 - a. le fichier /tmp/MACHINES contiendra :
 - i. permissions : 444
 - ii. propriétaire: user1
 - iii. groupe propriétaire : user1
 - b. vérifier les permissions, le propriétaire et le groupe propriétaire du fichier /tmp/MACHINES
5. Supprimer le fichier /tmp/MACHINES à l'aide du **module file**.
6. Installer la dernière version de git sur les hôtes avec le **module apt avec l'option a**
7. Faire un update du cache apt sur tous les noeuds

CLI Ansible : Pratique 1 (réponses)



1. `cat /etc/ansible/hosts (ansible --list-hosts all)`
2. `ansible all -m command -a "free -h"`
3. `ansible all -m shell -a "date; sleep 5; date"`
 - a. `ansible all -b -m group -a "name=user1 state=present"`
 - b. `ansible all -b -m user -a "name=user1 group=user1 createhome=yes"`
 - c. `ansible all -b -m copy -a "src=/etc/hosts dest=/tmp/MACHINES mode=444 owner=user1 group=user1"`
 - d. `ansible all -m command -a "ls -l /tmp/MACHINES"`
1. `ansible all -b -m ansible.builtin.file -a "dest=/tmp/MACHINES state=absent"`
2. `ansible all -b -m apt -a "name=git state=latest"`
3. `ansible all -b -m apt -a "update_cache=true"`

Inventory Ansible



Inventory Ansible



inventory = inventaire des machines **et** de leurs variables , puis ensuite on a les `host_vars` et les `groups_vars` qui sont les variables d'inventaire

élément essentiel car il décrit votre infra :

- * vos serveurs
- * vos types de serveurs : ex. web server, database, server de backup...

nécessaire pour travailler proprement

deux types d'instances ou objets :

- * `hosts` (servers à l'échelle individuelle)
- * `groupes` (groupement de `hosts` où on va pouvoir appliquer des patterns de `hosts`)

Inventory Ansible



plusieurs formats :

- * ini = plat (pas le plus intéressant, pas de structure en arbre visuelle, le seul qui peut prendre ce format)
- * yaml = plus homogène (tout le reste est en yaml, et souvent plugin yaml dans les IDE, structure arbre visuelle)
- * json = pour manipuler (utiliser des outils annexes par exemple avec python)

possibilité d'utiliser des patterns (pratique selon une nomenclature de servers bien établie)

inventory =

- * fichier d'inventaire
- * répertoire group_vars
- * répertoire host_vars

Inventory Ansible



fichier d'inventaire :

groupe racine \Rightarrow all

groupes enfants

exemple : ici parent et enfant sont des groupes

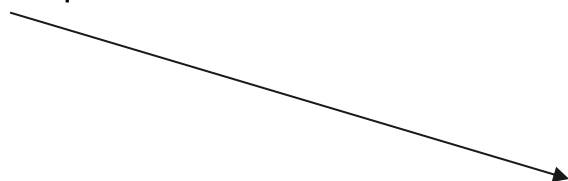
- * un groupe parent1
- * groupes enfants : enfant1 et enfant2
- * "sous" enfant de enfant2 : enfant3
- * enfant1 = srv1 et srv2
- * enfant2 = srv3
- * parent1 = srv4
- * enfant3= srv5

Inventory Ansible



format ini : je dis dans quel groupe je mets quel server
dans la moitié 1, dans la seconde moitié on répartit les
groupes

La structure de l'arbre ne vient qu'à la fin du fichier



```
[parent1]
srv4
[enfant1]
srv1
srv2
[enfant2]
srv3
[enfant3]
srv5
[parent1:children]
enfant1
enfant2
[enfant2:children]
enfant3
```

Inventory Ansible



format yaml :

```
all:
  children:
    parent1:
      hosts:
        srv4:
      children:
        enfant1:
          hosts:
            srv1:
            srv2:
        enfant2:
          hosts:
            srv3:
    children :
      enfant3:
        hosts:
          srv5:
```

Inventory Ansible



format yaml :

On peut aussi passer un groupe à un autre groupe : utiliser des groupes qui font appel à des mêmes serveurs, ici on déclare les hosts de parent 2 faisant partie de parent 1. Ici le groupe parent2 est au même niveau que le groupe parent1.

```
all:
  children:
    parent1:
      parent2:
        hosts:
          srv4:
        children:
          enfant1:
            hosts:
              srv1:
              srv2:
          enfant2:
            hosts:
              srv3:
            children:
              enfant3:
                hosts:
                  srv5:
    parent2:
      hosts:
        srv6:
        srv7:
        srv8:
        srv9:
```

Inventory Ansible



format yaml : utilisation de patterns

(un peu comme la syntaxe des listes en python)

```
all:
  children:
    parent1:
      parent2:
        hosts:
          srv4:
            children:
              enfant1:
                hosts:
                  srv[1:2]:
              enfant2:
                hosts:
                  srv3:
                children:
                  enfant3:
                    hosts:
                      srv5:
            parent2:
              hosts:
                srv[6:]:
```

Inventory Ansible



un peu plus vers la pratique

- * couche commune > common : par exemple le déploiement d'une stack commune à tous mes servers, donc on crée un sous-groupe qui va s'appeler common, qui est le groupe racine à tous. Et on va lui créer des groupes enfants,
- * serveurs web nginx > webserver
- * bases de données > dbserver
- * applications dockerisées ou non > app / appdock
- * puis monitoring qui lié à toutes les machines users > monitoring, comme ça par exemple si j'ai besoin par la suite de monitorer d'autres éléments il me suffira des les rajouter dans le groupe common

Format json : https://linuxhint.com/ansible_inventory_json_format/

Inventory Ansible

* couche commune > common : par exemple le déploiement d'une stack commune à tous mes servers,

donc on crée un sous-groupe qui va s'appeler common, qui est le groupe racine à tous. Et on va lui créer des groupes enfants,

- * serveurs web nginx > webserver
- * bases de données > dbserver
- * applications dockerisées ou non > app / appdock
- * puis monitoring qui lié à toutes les machines users >

monitoring, comme ça par exemple si j'ai besoin par la suite de monitorer d'autres éléments sans les mettre dans common il me suffira des les rajouter dans le groupe children juste sous all

```
all:
  children:
    common:
      children:
        webserver:
          hosts:
            srv[1:4]:
        dbserver:
          hosts:
            srv[5:6]:
        app:
          hosts:
            srv[7:10]:
        appdock:
          hosts:
            srv[11:15]:
  monitoring:
    children:
      common:
```

Inventory Ansible : variables



ansible = forte notion de précédence des variables (elles ont un ordre hiérarchique)

On peut les regrouper en 4 familles :

- * Configuration settings

- * Command-line options > VARIABLES D'INVENTAIRES (on a vu l'option -e

pour spécifier des variables en ligne de commande, -i pour inventaire ...)

- * Playbook keywords au niveau des rôles

- * Variables

il existe environ 22 niveaux de variables

Inventory Ansible : variables



environ 22 types: de la plus faible à
la plus forte

En connaître 7 ou 8 est déjà bien

Doc :

https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html

!

```
command line values (eg "-u user")
role defaults [1]
inventory file or script group vars [2]
inventory group_vars/all [3]
playbook group_vars/all [3]
inventory group_vars/* [3]
playbook group_vars/* [3]
inventory file or script host vars [2]
inventory host_vars/* [3]
playbook host_vars/* [3]
host facts / cached set_facts [4]
play vars
play vars_prompt
play vars_files
role vars (defined in role/vars/main.yml)
block vars (only for tasks in block)
task vars (only for the task)
include_vars
set_facts / registered vars
role (and include_role) params
include params
extra vars (always win precedence) (ex : -e en cli)
```


Inventory Ansible : variables

On va tester nos variables d'inventaire

Variables d'inventaire :

- fichier d'inventaire
- group_vars (répertoire)
- host_vars (répertoire)

Squelette de base de notre fichier inventory :

```
├── 00_inventory.yml
├── group_vars
│   ├── all.yml
│   ├── dbserver.yml
│   └── webserver
│       ├── vault.yml
│       └── webserver.yml
└── host_vars
    ├── srv1
    │   └── srv1.yml
    └── srv2.yml
```

Inventory Ansible : variables

multienv >> On crée souvent un répertoire env, et dans chaque environnement on va faire la même structure, on pourra bien sûr la faire évoluer. ça va nous permettre de bien isoler les choses. Puis quand on fera une commande ansible il nous suffira de passer une commande avec l'option -i env/dev ou -i env/prod ... pour récupérer l'inventaire avec les variables qui y correspondent

```
├─ dev
│   ├── 00_inventory.yml
│   ├── group_vars
│   │   ├── all.yml
│   │   ├── dbserver.yml
│   │   └── webserver
│   │       ├── vault.yml
│   │       └── webserver.yml
│   └── host_vars
│       ├── srv1
│       │   └── srv1.yml
│       └── srv2.yml
├─ prod
│   ├── 00_inventory.yml
│   ├── group_vars
│   │   ├── all.yml
│   │   ├── dbserver.yml
│   │   └── webserver
│   │       ├── vault.yml
│   │       └── webserver.yml
│   └── host_vars
│       ├── srv1
│       │   └── srv1.yml
│       └── srv2.yml
└─ stage
    ├── 00_inventory.yml
    ├── group_vars
    │   ├── all.yml
    │   ├── dbserver.yml
    │   └── webserver
    │       ├── vault.yml
    │       └── webserver.yml
    └── host_vars
        ├── srv1
        │   └── srv1.yml
        └── srv2.yml
```

Inventory Ansible : variables

On va utiliser une commande de test :

```
ansible -i "node2," all -e "var1=bonjour" -m debug -a 'msg={{ var1 }}'
```

là avec l'option -e on a un des niveaux les plus élevés
pour les settings de variables

Créer un dossier env , à l'intérieur créer un fichier 00_inventory.yml

Deux sous-groupes : common et monitoring

Dans le groupe common; deux sous-groupes : webserver et dbserver

```
all:
  children:
    common:
      children:
        webserver:
          hosts:
            node[2:3]:
          vars:
            var1: "webserver"
        dbserver:
          hosts:
            node4:
            node5:
              var1: "node5"
          vars:
            var1: "dbserver"
    monitoring:
      children:
        webserver:
        dbserver:
```

Inventory Ansible : variables

```
ansible -i 00_inventory.yml all -e "var1=bonjour" -m debug -a 'msg={{ var1 }}'
```

On observe que le setting avec l'option e a surpassé le settings dans le yml. Il l'emporte sur toutes les autres.

et si on relance en enlevant l'option e

```
ansible -i 00_inventory.yml all -m debug -a 'msg={{ var1 }}'
```

```
all:
  children:
    common:
      children:
        webserver:
          hosts:
            node[2:3]:
          vars:
            var1: "webserver"
        dbserver:
          hosts:
            node4:
            node5:
              var1: "node5"
          vars:
            var1: "dbserver"
      monitoring:
        children:
          webserver:
          dbserver:
```

Inventory Ansible : variables

Cependant, l'idéal n'est pas de fonctionner comme cela : ajouter trop de variables risque par la suite de rendre le fichier illisible, trop brouillon.

```
tree
```

```
mkdir group_vars
```

```
vim group_vars/webserver.yml
```

```
var1: "gp_webserver"
```

```
vim group_vars/dbserver.yml
```

```
var1: "gp_dbserver"
```

```
ansible -i 00_inventory.yml all -m debug -a 'msg={{ var1 }}'
```

```
all:
  children:
    common:
      children:
        webserver:
          hosts:
            node[2:3]:
        dbserver:
          hosts:
            node4:
            node5:
              var1: "node5"
```

Inventory Ansible : variables

ou encore quand on a beaucoup de variables, on a plutôt tendance à grouper les variables à l'intérieur du group_vars, et dans ce cas la reconnaissance d'ansible se fera par le nom du répertoire. Donc peu importe le nom du fichier, il faut par exemple que le répertoire ici soit bien nommé dbserver

```
├── group_vars
│   ├── all.yml
│   ├── dbserver
│   │   └── dbserver.yml
│   └── webserver
│       └── webserver.yml
├── host_vars
│   ├── node2
│   │   └── variables.yml
│   └── node5.yml
└── inventory.yml
```

```
all:
  children:
    common:
      children:
        webserver:
          hosts:
            node[2:3]:
        dbserver:
          hosts:
            node4:
            node5:
              var1: "node5"
```

Un outil pour l'inventaire :

Utiliser des scripts pour automatiser certains éléments repris de l'inventaire

```
ansible-inventory
```

Export au format json par défaut (serveurs et variables d'inventaire)

```
ansible-inventory -i <inventory_file> --list
ansible-inventory -i <inventory_file> --list --yaml
```

Une partie avec les valeurs de variables pour chacun des hosts (intéressant pour déboguer : quelle valeur, dans quel groupe) : utilise les valeurs dans group_vars

```
ansible-inventory -i 00_inventory.yml --list --yaml
```

```
ansible-inventory -i 00_inventory.yml --list --export
```

 (répartition variables directement avec la structure)

```
all:
  children:
    common:
      children:
        webserver:
          hosts:
            node[2:3]:
        dbserver:
          hosts:
            node4:
            node5:
      nocommon:
        hosts:
          node6:
```

Un outil pour l'inventaire :

```
ansible-inventory -i 00_inventory.yml --graph
```

```
ansible-inventory -i 00_inventory.yml --graph --vars
```

Export vers un fichier

```
ansible-inventory -i 00_inventory.yml --list --yaml --output myinv.txt
```

```
ansible-inventory -i 00_inventory.yml --graph --vars --output myinv.txt
```

```
all:
  children:
    common:
      children:
        webserver:
          hosts:
            node[2:3]:
        dbserver:
          hosts:
            node4:
            node5:
      nocommon:
        hosts:
          node6:
```


Un outil pour l'inventaire :

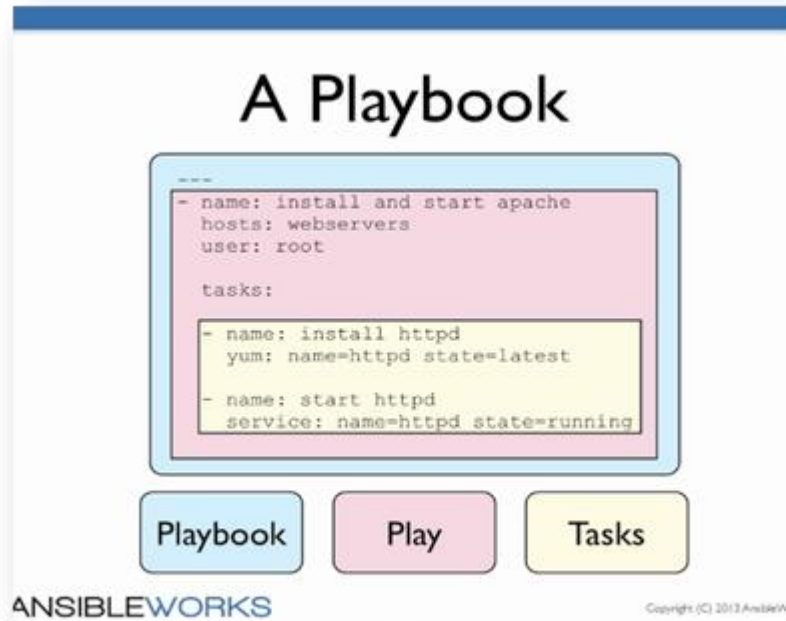


```
pip3 install toml
```

```
ansible-inventory -i 00_inventory.yml --list --toml --output myinv.txt
```

```
all:
  children:
    common:
      children:
        webserver:
          hosts:
            node[2:3]:
        dbserver:
          hosts:
            node4:
            node5:
    nocommon:
      hosts:
        node6:
```

Ansible Playbook



Playbook : début



- * fichier déclenchant les actions à réaliser
- * sert à articuler l'inventary (les groupes) avec les rôles (répertorie les tasks, jamais un nom de serveur dans un rôle)
- * peut inclure des tasks (actions) > éviter (question d'organisation)
- * peut inclure des variables (éviter autant que possible, problèmes de maintenance)
- * peut faire tout ce que fait un rôle (globalement) > rôle (mais jamais mettre une task dans un playbook)
- * spécifier quel user et comment ? (ex: élévation de privilèges)

une commande : **ansible-playbook**

Playbook : nombreuses options



- -i : inventory
- -l : limit > spécifier des noms de groupes ou serveurs ou patterns
- -u : user
- -b : become > sudo
- -k : prompt pour password de ssh (à éviter)
- -K : password d'élévation de privilèges (ex: pour install)
- -C : check > dry run (lancer ansible playbook à vide, certaines actions bloquent comme création répertoire puis utilisation de ce répertoire)
- -D : diff > afficher les différences avant/après les tasks (cumuler -C et -D)
- --ask-vault : affiche un prompt pour le password vault de chiffrement
- --vault-password-file : stocker le vault password dans un fichier (évite de saisir le password sur le prompt et fait passer directement le fichier)
- --syntax-check : vérifie la syntaxe
- -e : surcharge n'importe quelle variable

Playbook : nombreuses options



- -f : nombre de parallélisation
- -t : filtrer les tags (pour ne pas rejouer l'intégralité du playbook)
- --flush-cache : recollecter les gather-facts (et pas se baser sur des anciens)
- --step : confirmation via le prompt pour chaque task
- --start-at-task : commencer à une tâche particulière
- --list-tasks : quelles tâches seront exécutées (à faire avant la commande précédente)
- --list-tags : même chose pour les tags

Un outil en CLI pour vérifier la qualité de vos yaml : YAMLLINT

```
sudo apt install yamllint
```

Puis appeler le fichier concerné :

```
yamllint .
```

Playbook : premier

```
sudo vim playbook.yml
```

```
- name: Premier Playbook
  hosts: all
  remote_user: vagrant
  tasks:
    - name: je debug
      debug:
        msg: "Hello {{ var1 }}"
```

00_inventory.yml

```
all:
  children:
    webserver:
      hosts:
        node[2:3]:
    dbserver:
      hosts:
        node[4:5]:
```

- Si caractères spéciaux pour le nom du playbook, utiliser ""
- hosts : sur quelle machine on va l'exécuter
conditions possibles dans le playbook (à éviter, mais surtout pas dans les rôles)

Playbook : premier



Remarque : le playbook n'est pas là pour gérer les environnements, c'est l'inventory qui s'en occupe (inventory de dev, de prod...)

```
ansible-playbook -i 00_inventory.yml playbook.yml
```

-k : pour demander le mot de passe ssh

Mais pbe : Ansible dans ses dernières versions n'accepte pas le password ssh directement → besoin de sshpass

```
sudo apt install sshpass
```

```
sshpass -p "vagrant" ansible-playbook -i 00_inventory.yml -k playbook.yml
```

Playbook : premier



Il est possible parfois d'avoir une erreur avec le fingerprint pour les hosts (StrictHostKeys)

```
sudo vim /etc/ansible/ansible.cfg
```

```
# uncomment this to disable SSH key host checking  
#host_key_checking = False
```



```
# uncomment this to disable SSH key host checking  
host_key_checking = False
```


Playbook : premier



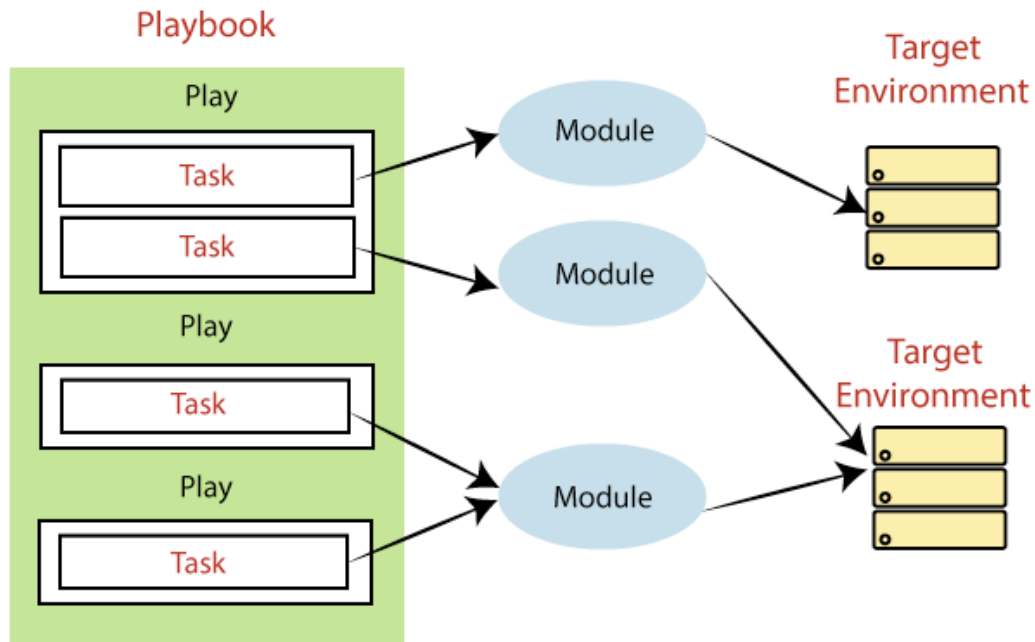
Possibilité de passer des variables dans le playbook mais mauvaise pratique :

```
- name: Premier Playbook
  hosts: all
  remote_user: vagrant
  vars:
    var1: "playbook"
  tasks:
    - name: je debug
      debug:
        msg: "Hello {{ var1 }}"
```

Précédence des variables dans le playbook sur celle dans l'inventary et encore plus avec l'option -e :

```
sshpass -p "vagrant" ansible-playbook -e "var1=plus_forte" -i 00_inventory.yml -k playbook.yml
```

Ansible Modules



Module file



Doc : https://docs.ansible.com/ansible/latest/modules/file_module.html

Commande sur pc: `ansible-doc file`

Objectifs : gestion des fichiers et répertoires

Périmètre : fichiers, répertoires et liens symboliques

Options courantes :

- **attribute** : paramètres particuliers d'un fichier : ex immutabilité (empêcher qu'un fichier soit supprimé même par un root)
- **force** : pour les liens symboliques (si le fichier source n'existe pas, la destination existe alors que de base linux le fait même sans le fichier)
- **group/owner** : propriétaire et groupe de l'élément
- **mode** : sous les deux formats 0755 ou u=rwx, g=rx, o=rx
- **path** : localisation
- **recurse** : comme `mkdir -p`
- **src** : pour les liens soft (link vers le fichier) ou hard (link vers l'inode)

Module file



Autres Options :

- **state** : (très utilisé) **type** : absent/directory/file/hard/link/touch
 - absent : si présent il faut le supprimer
 - file : vérifie existence (vérifie l'idempotence)
 - touch: crée le fichier vide

Inventaire

```
all:
  children:
    common:
      hosts:
        node2:
```

Seconde fenêtre avec node2

```
sudo vim playbook.yml
```

```
- name: Premier Playbook
  hosts: common
  tasks:
    - name: check connexion
      ping:
```

ping permet aussi de récupérer les gather_facts

Module file

Création d'un répertoire :

```
- name: création du répertoire /tmp/moi
  ansible.builtin.file:
    path: /tmp/moi/
    state: directory
```

Vérifier sur les hosts la présence du directory avec `ls -la` pour voir quel user a les droits sur ce

```
- name: création du répertoire /tmp/moi
  ansible.builtin.file:
    path: /tmp/moi/
    state: directory
    owner: root
```

```
ansible-playbook -i 00_inventory.yml -u vagrant -k playbook.yml
```

Mais ici échec car il demande un sudo

Module file

Plusieurs solutions :

depuis la ligne de commande avec option b

```
sshpass -p ansible-playbook -i 00_inventory.yml -u vagrant -k -b playbook.yml
```

depuis le playbook

```
- name: Premier Playbook
  hosts: common
  tasks:
    - name: check connexion
      ping:
    - name: création du répertoire /tmp/moi
      file:
        path: /tmp/moi/
        state: directory
        owner: root
      become: yes
```

```
- name: Premier Playbook
  hosts: common
  become: yes
  tasks:
    - name: check connexion
      ping:
    - name: création du répertoire /tmp/moi
      file:
        path: /tmp/moi/
        state: directory
        owner: root
```

Module file



Ici il ne demande pas le password du sudo car le user vagrant est avec nopassword

```
stat /moi
```

sur le node2 :

Rq: idempotence

Si Ansible rejoue plusieurs fois le même playbook et n'a pas besoin de modifier quelque chose, il n'exécute pas la tâche et affiche seulement ok

```
- name: Premier Playbook
  hosts: common
  become: yes
  tasks:
    - name: check connexion
      ping:
    - name: création du répertoire /tmp/moi
      file:
        path: /tmp/moi/
        state: directory
        owner: root
        group: root
        mode: 0755
```

Module file



Mode recurse possible
sur le state directory

```
- name: Premier Playbook
  hosts: common
  become: yes
  tasks:
    - name: check connexion
      ping:
    - name: création du répertoire /tmp/moi/1/2/3/4/5
      file:
        path: /tmp/moi/1/2/3/4/5
        recurse: yes
        state: directory
        owner: root
        group: root
        mode: 0755
```


Module file



Création d'un fichier

```
- name: Premier Playbook
  hosts: common
  become: yes
  tasks:
    - name: check connexion
      ping:
    - name: création du fichier /tmp/moi/1/2/3/4/5/fichier.txt
      file:
        path: /tmp/moi/1/2/3/4/5/fichier.txt
        state: touch
        owner: root
        group: root
        mode: 0755
```

Module file



Vérification de l'existence d'un
fichier avec les droits demandés

```
- name: Premier Playbook
  hosts: common
  become: yes
  tasks:
    - name: check connexion
      ping:
    - name: création du répertoire /tmp/moi/1/2/3/4/5/fichier.txt
      file:
        path: /tmp/moi/1/2/3/4/5/fichier.txt
        state: file
        owner: root
        group: root
        mode: 0755
```

Module file



Supprimer les éléments

```
- name: Premier Playbook
  hosts: common
  become: yes
  tasks:
    - name: check connexion
      ping:
    - name: suppression du répertoire /tmp/moi/
      file:
        path: /tmp/moi/
        state: absent
```

Module User



Doc : https://docs.ansible.com/ansible/latest/collections/ansible/builtin/file_module.html

Commande : `ansible-doc user`

Equivalence : `useradd/adduser/userdel/deluser/luseradd`

→ PARAMÈTRES

- **append** : `yes/no` > en lien avec `groups` / ajout aux groupes ou changement
 - `yes` : n'écrase pas
 - `no` : écrase
- **comment** : commentaire associé au user
- **create_home** : création ou non de la home (par défaut à `yes`)
- **expires**: date d'expiration d'un user au format epoch
- **force**: permet de forcer la suppression des fichiers d'un user dans sa home
- **generate_ssh_key** : génère une clef ssh pour l'utilisateur (en localhost, diffusion sur hosts après)

```
date "+%s" -d "10/06/2040 10:00:00"
```

Module User



→ PARAMÈTRES (SUITE)

- **group** : définit le groupe principal de l'utilisateur (par défaut c'est le nom du user)
- **groups**: définit les groupes secondaires qui seront ajoutés (sudo, docker...)
- **home**: définition de la home du user avec un chemin désiré (≠ create_home)
- **name**: nom utilisateur
- **password**: hash du password (à nous de définir le hash)
- **password_lock** : verrouiller le password du user
- **remove**: avec le state absent qui supprime le user, remove supprime en même temps les répertoires du user (sa home)
- **shell**: shell par défaut du user (bash, zsh...)
- **skeleton**: avec create_home, pour définir le squelette à appliquer
- **ssh_key_passphrase**: définit la passphrase de la clef ssh sinon pas de passphrase
- **ssh_key_file**: chemin de la clef ssh
- **ssh_key_type**: rsa par défaut
- **state**: present ou absent
- **system**: définir un compte system ou non à la création

Module User : Exemples

Création d'un user avec password :

00_inventory.yml

```
all:
  vars:
    ansible_python_interpreter:
      /usr/bin/python3
  hosts:
    192.168.10.11:
    192.168.10.12:
    192.168.10.13:
```

id devops

playbook.yml

```
- name: Playbook User
  hosts: all
  become: yes
  tasks:
    - name: création du user devops
      user:
        name: devops
        state: present
        password: "{{ 'password' | password_hash('sha512') }}"
```

Module User : Exemples

Ajout du user devops dans un groupe secondaire:

playbook.yml

00_inventory.yml

```
all:
  vars:
    ansible_python_interpreter:
/usr/bin/python3
  hosts:
    192.168.10.11:
    192.168.10.12:
    192.168.10.13:
```

```
- name: Playbook User
  hosts: all
  become: yes
  tasks:
    - name: création du user devops
      user:
        name: devops
        state: present
        groups: sudo
        password: "{{ 'password' | password_hash('sha512') }}"
```

id devops

sudo -l

ou depuis node1
:

ssh devops@192.168.15.XX

Module User : Exemples

Changement de l'UID du user devops:

playbook.yml

00_inventory.yml

```
all:
  vars:
    ansible_python_interpreter:
      /usr/bin/python3
  hosts:
    192.168.10.11:
    192.168.10.12:
    192.168.10.13:
```

```
- name: Playbook User
  hosts: all
  become: yes
  tasks:
    - name: création du user devops
      user:
        name: devops
        state: present
        uid: 1200
        groups: sudo
        password: "{{ 'password' | password_hash('sha512') }}"
```

`id devops`

`sudo -l`

ou depuis node1
:

`ssh devops@192.168.15.XX`

Module User : Exemples



Remarque :

➤ `generate_ssh_key`

Pas beaucoup d'intérêt

Il vaut mieux générer les clés depuis le node manager et ensuite gérer les

nodes avec

Garder la clé privée dans le node manager

Module User : Exemples

playbook.yml

Génération de clefs ssh avec
récupération de l'output:

00_inventory.yml

```
all:
  vars:
    ansible_python_interpreter:
      /usr/bin/python3
  hosts:
    192.168.10.11:
    192.168.10.12:
    192.168.10.13:
```

```
- name: Playbook User
  hosts: all
  become: yes
  tasks:
    - name: création du user devops
      user:
        name: devops
        state: present
        generate_ssh_key: yes
        password: "{{ 'password' | password_hash('sha512') }}"
        register: __user_devops
    - name: debug
      debug:
        var: __user_devops
```

```
cd /home/devops/.ssh
```

```
ls -la
```

Module User : Exemples

playbook.yml

Supprimer le user sans le home :

00_inventory.yml

```
all:
  vars:
    ansible_python_interpreter:
      /usr/bin/python3
  hosts:
    192.168.10.11:
    192.168.10.12:
    192.168.10.13:
```

```
cd /home/devops/
```

```
id devops
```

```
- name: Playbook User
  hosts: all
  become: yes
  tasks:
    - name: création du user devops
      user:
        name: devops
        state: present
        generate_ssh_key: yes
        password: "{{ 'password' | password_hash('sha512') }}"
        register: __user_devops
    - name: debug
      debug:
        var: __user_devops
    - name: remove devops
      user:
        name: devops
        state: absent
        remove: yes
```

Module APT : Paramètres



- Documentation:

https://docs.ansible.com/ansible/latest/collections/ansible/builtin/copy_module.html

- Objectifs : copier des fichiers ou du contenu
- Equivalent ? scp

PARAMÈTRES :

- * attributes : attributs du fichier
- * backup : réalise une copie datée avant la copie
- * checksum : vérification du fichier via un hash
- * content : dans le cas où la source n'est pas un fichier mais une variable ou un string
- * decrypt : déchiffre les fichiers si ils sont vaultés (défaut : yes)
- * dest : localisation du fichier sur les serveurs target
- * directory_mode : dans le cas d'une recopie en mode récursif

Module APT : Paramètres (suite)



- **install_recommends** : activer ou désactiver les paquets recommandés (dépend des OS)
- **name** : nom du paquet
- **only_upgrade** : met à jour uniquement les paquets installés
- **policy_rc_d** : règle de déclenchement automatique à l'installation d'un paquet
- **purge** : purge les fichiers de configurations (--purge)
- **state** : present / absent / latest / present / build-dep
- **update_cache** : réaliser un update avant l'installation
- **update_cache_retries** : nombre de tentatives de l'update
- **update_cache_retry_max_delay** : délai de chaque retry
- **upgrade** : yes / no / safe / dist / full

Module APT : Mise à jour du cache



playbook.yml

00_inventory.yml

```
all:
  vars:
    ansible_python_interpreter:
      /usr/bin/python3
  hosts:
    192.168.10.11:
    192.168.10.12:
    192.168.10.13:
```

```
- name: Playbook User
  hosts: all
  become: yes
  tasks:
    - name: nettoyage du cache avec module apt
      apt:
        update_cache: yes
        cache_valid_time: 3600
```

Module APT : Installation d'un paquet

00_inventory.yml

```
all:
  vars:
    ansible_python_interpreter:
      /usr/bin/python3
  hosts:
    192.168.10.11:
    192.168.10.12:
    192.168.10.13:
```

```
apt list -i haproxy
```

playbook.yml

```
- name: Playbook User
  hosts: all
  become: yes
  tasks:
    - name: installation d'haproxy
      apt:
        name: haproxy
        state: latest
```

Possibilité ajout **default_release** sous **name** à la place de **state** pour choisir version du paquet
Attention à **latest** : la recherche de nouvelles versions et une éventuelle install (voir impacts)

Module APT : Suppression d'un paquet

playbook.yml

00_inventory.yml

```
all:
  vars:
    ansible_python_interpreter:
      /usr/bin/python3
  hosts:
    192.168.10.11:
    192.168.10.12:
    192.168.10.13:
```

```
apt list -i haproxy
```

```
- name: Playbook User
  hosts: all
  become: yes
  tasks:
    - name: suppression d'haproxy
      apt:
        name: haproxy
        state: absent
```

Pour effacer complètement fichiers et dépendances en plus :

```
purge: yes
```

```
autoremove: yes
```


Module COPY : Paramètres



Doc: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/copy_module.html

Objectifs : copier des fichiers ou du contenu

Equivalent à une commande à base de scp

PARAMÈTRES :

- **attributes** : attributs du fichier (éviter que le fichier soit supprimable par ex)
- **backup** : réalise une copie datée avant la copie (permet des rollback plus sécurisé)
- **checksum** : vérification du fichier via un hash
- **content** : dans le cas où la source n'est pas un fichier mais une variable ou un string
- **decrypt** : déchiffre les fichiers s'ils sont vaultés (défaut : yes)
- **dest** : localisation du fichier sur les serveurs target
- **directory_mode** : dans le cas d'une recopie en mode récursif

Module COPY : Paramètres (suite)



- **follow** : indiquer le filesystem dans la destination
- **force** : remplace le fichier s'il est différent de la source (yes par défaut)
- **group** : group propriétaire du répertoire ou du fichier
- **local_follow** : indique le filesystem dans la source
- **mode** : permissions du fichier ou du répertoire (0755, u+rw, g+rx, o+rx)
- **owner** : user propriétaire
- **remote_src** : no > copie du master vers la target, yes > copie de la target vers la target (au sein du même serveur)
- **src** : localisation de la source
 - **attention** : roles / dir files / .
- **validate** : commande jouée pour valider le fichier avant de le copier (le fichier se situe %s)

Module COPY : Envoi d'un fichier

```
- name: Découverte module COPY
  hosts: all
  become: yes
  tasks:
    - name: copy
      copy:
        src: test.txt
        dest: /tmp/test_second.txt
```

Par défaut, Ansible va chercher le fichier à trois endroits différents :

- dans un dossier nommé files
- de l'endroit où le playbook est lancé
- soit dans le répertoire des rôles (files) si on utilise des rôles

Si on ne donne pas de nom au fichier dest, il reprendra le même que celui en src

Modifier le fichier test.txt depuis le manager et relancer le playbook (change apparu

Re-modifier le fichier source et ajouter **force: no** (ne modifie pas, vérifie juste la présence de l'élément)

Module COPY : Création d'un répertoire (mode récursif)



```
mkdir -p tmp/essai/{1,2,3}
```

```
mv test.txt /tmp/essai/1/
```

```
- name: Découverte module COPY
  hosts: all
  become: yes
  tasks:
    - name: copy
      copy:
        src: tmp/
        dest: /tmp/
```

Module COPY : remote_src



```
- name: Découverte module COPY
hosts: all
become: yes
tasks:
  - name: copy
    copy:
      src: tmp/essai/
      dest: /home/node2
      remote_src: yes
```

Module COPY : boucle avec with_items



```
sudo vim test1.txt
```

```
sudo vim test2.txt
```

Module COPY : boucle avec with_items

```
- name: Variables et Boucle avec COPY
  hosts: all
  become: yes
  vars:
    mesfichiers:
      - { source: "test1.txt", destination: "/tmp/test1.txt", mode: "0755", owner: "vagrant" }
      - { source: "test2.txt", destination: "/home/vagrant/test2.txt", mode: "0644", owner:
"vagrant" }
  tasks:
    - name: copy
      copy:
        src: "{{ item.source }}"
        dest: "{{ item.destination }}"
        mode: "{{ item.mode }}"
        owner: "{{ item.owner }}"
      with_items:
        - "{{ mesfichiers }}"
```

Module COPY : auto-découverte de fichiers avec pattern



```
- name: Patterns avec COPY
hosts: all
become: yes
tasks:
  - name: copy
    copy:
      src: "{{ item }}"
      dest: /tmp
    with_fileglob:
      - test*
```


Module COPY : back-up avant modification



Modifier le fichier test1.txt

backup permet de n'agir que sur les modifications et création d'un fichier horodaté qui est le backup

```
- name: Patterns avec COPY
  hosts: all
  become: yes
  tasks:
    - name: copy
      copy:
        src: "{{ item }}"
        dest: /tmp
        backup: yes
      with_fileglob:
        - test*
```

Module COPY : injecter directement du contenu



content à la place de src

Le pipe | permet de passer une variable de plusieurs lignes

```
- name: Patterns avec COPY
hosts: all
become: yes
tasks:
  - name: copy
    copy:
      content: |
        Bonjour
        je suis sur la machine {{ ansible_host }}
      dest: /tmp/hello.txt
```

Module COPY : check avant la validation

Options : c pour check, f pour passer le file, %s pour récupérer le nom de fichier dans dest

```
- name: Variables et Boucle avec COPY
hosts: all
become: yes
tasks:
- name: Add devops user to the sudoers
  copy:
    dest: "/etc/sudoers.d/devops"
    content: "devops ALL=(ALL) NOPASSWD: ALL"
    owner: root
    group: root
    mode: 0600
    validate: visudo /usr/sbin/visudo -cf %s
```

```
sudo cat /etc/sudoers.d/devops
```

Module COPY : check avant la validation



Remarque : si une erreur est dans le playbook par exemple au niveau du content, il ne l'exécute pas → confort important

```
- name: Variables et Boucle avec COPY
hosts: all
become: yes
tasks:
  - name: Add devops user to the sudoers
    copy:
      dest: "/etc/sudoers.d/devops"
      content: "devops ALL=(ALL) AAAAA: ALL"
      owner: root
      group: root
      mode: 0600
      validate: visudo /usr/sbin/visudo -cf %s
```



Playbook : Pratique

Playbook : Pratique



A l'aide d'un seul playbook:

1. Créer pour le groupe webserver (2 nodes):
 - a. un user apacheadm dans le group users, admin
 - b. lui créer une home /home/apacheadm
 - c. Installer le paquet apache2

1. Créer pour le groupe dbserver (2 nodes):
 - a. un user utilisateur dans le groupe users
 - b. lui créer son home /home/utilisateur
 - c. Lui créer un répertoire /opt/oracle avec mode 0755

Playbook : Pratique (réponse)



00_inventory.yml

```
all:
  vars:

ansible_python_interpre
ter: /usr/bin/python3
  children:
    webservers:
      hosts:
        node[2:3]
    appservers:
      hosts:
        node[4:5]
```

/etc/hosts

```
127.0.0.1      localhost

# The following lines are desirable for IPv6 capable hosts
::1           ip6-localhost  ip6-loopback
fe00::0       ip6-localnet
ff00::0       ip6-mcastprefix
ff02::1       ip6-allnodes
ff02::2       ip6-allrouters
ff02::3       ip6-allhosts
127.0.1.1      ubuntu-focal  ubuntu-focal

127.0.2.1 node1 node1
192.168.15.10 node1
192.168.15.11 node2
192.168.15.12 node3
192.168.15.13 node4
192.168.15.14 node5
~
```

Playbook : Pratique (réponse)

```
---
# Play1 - WebServer related tasks
- name: Play Web - Create apache directories and username in web
servers
  hosts: webservers
  become: true
  tasks:
    - name: Create group
      group:
        name: users
        state: present

    - name: Create group
      group:
        name: admin
        state: present

    - name: create username apacheadm
      user:
        name: apacheadm
        groups: users,admin
        append: yes
        home: /home/apacheadm

    - name: install apache2
      apt:
        name: apache2
        state: latest
```

Dans un seul playbook

```
# Play2 - Application Server related tasks
- name: Play app - Create utilisateur directories and
username in app servers
  hosts: appservers
  become: yes
  tasks:
    - name: Create a username for utilisateur
      user:
        name: utilisateur
        group: users
        append: yes
        home: /home/tomcat

    - name: create a directory for apache tomcat
      file:
        path: /opt/oracle
        owner: utilisateur
        group: users
        state: directory
        mode: 0755
```


Ansible Templating



Jinja

Jinja : Outil de templating

Nous allons aborder quelques possibilités avec l'outil Jinja de templating de python

Ressemblance avec le templating go, mais reste différent

Déclaration d'une variable en format yaml :

```
my_var: "valeur"
```



```
{{ my_var }}
```

Il faut réussir à trouver à équilibre entre Jinja et Ansible : l'un ne doit pas être trop complexe par rapport à l'autre → Viser un code facilement maintenable

Jinja est très utile pour des boucles, des conditions (notamment avec des types de variables)

Jinja : les boucles

Déclaration d'une liste en format yaml :

```
items:  
- valeur1  
- valeur2  
- valeur3
```

Nous allons mettre en place deux tasks : **debug** pour afficher à l'écran et **copy** pour envoyer du contenu

Rq : double accolade uniquement pour les variables

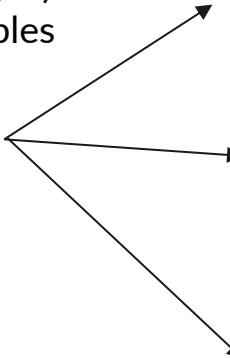
```
- name: jinja3  
  hosts: all  
  
  vars:  
    items:  
      - valeur1  
      - valeur2  
      - valeur3  
  
  tasks:  
    - name: debug  
      debug:  
        msg: |  
          {% for item in items %}  
            {{ item }}  
          {% endfor %}  
  
    - name: copy  
      copy:  
        content: |  
          {% for item in items %}  
            {{ item }}  
          {% endfor %}  
        dest: /tmp/test_jinja.txt
```

Jinja : les boucles

```
ansible-playbook -i 00_inventory.yml playbook.yml  
vagrant@node2:~$ cat /tmp/test_jinja.txt
```

Le debug affiche le résultat avec \n (retour à la ligne)
Et copy envoie le fichier avec les valeurs de variables

Pour éviter le retour à la ligne: trois possibilités :



```
content: |  
{% for item in items %}{{ item }}{% endfor %}
```

```
content: |  
{% for item in items %}  
{{ item -}}  
{% endfor %}
```

```
content: |  
{% for item in items %}  
{{ item }}  
{%- endfor %}
```

Jinja : les boucles

Ensuite, nous pouvons utiliser le dictionnaire, très pratique en déclaration d'infrastructure (IP, database de users...) , plus qu'une liste au format plat.

Rq: nous n'utilisons plus les tirets

```
- name: jinja3
  hosts: all

  vars:
    items:
      key1: valeur1
      key2: valeur2
      key3: valeur3

  tasks:
    - name: copy
      copy:
        content: |
          {% for item, data in items.items() %}
          {{ item }} >> {{ data }}
          {% endfor %}
        dest: /tmp/test_jinja.txt
```

Jinja : les boucles

On peut également faire une liste de dictionnaires

```
- name: jinja3
hosts: all

vars:
  items:
    - { key1: valeur1, key2: valeur2, key3:
valeur3 }
    - { key1: valeur4, key2: valeur5, key3:
valeur6 }
    - { key1: valeur7, key2: valeur8, key3:
valeur9 }
  tasks:
    - name: copy
      copy:
        content: |
          {% for item in items %}
          {{ item.key2 }} >> {{ item.key1 }}
          {% endfor %}
        dest: /tmp/test_jinja.txt
```

Jinja : les boucles

On peut également faire un dictionnaire de dictionnaires

Essayer avec `{{ items[item] }}`

On itère comme si c'était une liste

```
{% for item, datas in items.items() %}  
{{ item }} {{ datas.name }} {{ datas.city }}  
{% endfor %}
```

Donc les premiers éléments (ici itemx) se manipulent comme une liste avec ou sans tirets. Puis utilisation index ou la méthode items()

```
- name: jinja3  
hosts: all  
  
vars:  
  items:  
    item1:  
      name: jojo  
      city: toulouse  
    item2:  
      name: toto  
      city: jesaisplus  
  
tasks:  
- name: debug  
  debug:  
    msg: |  
      {% for item in items %}  
      {{ item }}  
      {% endfor %}  
  
- name: copy  
  copy:  
    content: |  
      {% for item in items %}  
      {{ items[item].name }} - {{ items[item].city }}  
      {% endfor %}  
  dest: /tmp/test_jinja.txt
```

Jinja : les boucles



On peut imbriquer deux boucles for

```
- name: jinja3
hosts: all

vars:
  items:
    item2:
      - titi
      - toto
      - tata
    item1:
      - tutu
      - tyty

tasks:
- name: copy
  copy:
    content: |
      {% for item, data in items.items() %}
      {% for d in data %}
      {{ d }}
      {% endfor %}
      {% endfor %}
  dest: /tmp/test_jinja.txt
```


Jinja : les boucles



Nous pouvons maintenant ordonner les clés.

Un pipe est appelé un filtre.

Ici dictsort filtre les premiers niveaux

La méthode items() est alors appliquée directement car c'est le filtre lui-même qui indique qu'il faut se comporter comme un dictionnaire

```
- name: jinja3
hosts: all

vars:
  items:
    item2:
      - titi
      - toto
      - tata
    item1:
      - tutu
      - tyty

tasks:
- name: copy
  copy:
    content: |
      {% for item, data in items | dictsort %}
      {% for d in data %}
      {{ d }}
      {% endfor %}
      {% endfor %}
  dest: /tmp/test_jinja.txt
```

Jinja : les conditions



Tout comme en python :

- Opérateurs de comparaison (==, !=, >=, <=, < ...)
- Opérateurs logiques (and/or/not)
- Boolean
- define / number / string / mapping (dict) / iterable...

Jinja : les conditions



Avec else et elif

```
- name: conditions avec jinja3
hosts: all

vars:
  var1: 1
  var2: 2

tasks:
  - name: copy
    copy:
      content: |
        {% if var1 < var2 %}
        var1 est inf à var2
        {% elif var1 == var2 %}
        var1 égale à var2
        {% else %}
        var1 est sup ou égale à var2
        {% endif %}
      dest: /tmp/test_jinja.txt
```

Jinja : les conditions



Avec un boolean

Pour seulement vérifier la présence
d'une variable : `{% if var1 %}`

Mais si variable commentée ou absente,
fait buger le script

```
- name: conditions avec jinja3
hosts: all

vars:
  #var1: 1
  var2: 2

tasks:
  - name: copy
    copy:
      content: |
        {% if var1 is defined %}
        var1 est définie
        {% else %}
        var1 n'est pas définie
        {% endif %}
    dest: /tmp/test_jinja.txt
```

Jinja : les conditions

Avec des opérateurs de comparaison

même chose qu'avec

```
{% if var1 != 1 %}
```

Possible avec des chaînes de caractères :

```
var1: "mysql"
```

```
{% if var1 != "mysql" %}
```

```
- name: conditions avec jinja3
  hosts: all

  vars:
    var1: 1
    var2: 2

  tasks:
    - name: copy
      copy:
        content: |
          {% if not var1== 1 %}
          var1 ok
          {% else %}
          var1 pas ok
          {% endif %}
        dest: /tmp/test_jinja.txt
```

Jinja : les conditions



Avec des opérateurs logiques :

```
- name: conditions avec jinja3
hosts: all

vars:
  var1: 1
  #var2: 2

tasks:
- name: copy
  copy:
    content: |
      {% if var1 is defined and var2 is defined %}
      variables ok
      {% elif var2 is not defined %}
      var1 ok
      var2 pas ok
      {% else %}
      variables pas ok
      {% endif %}
    dest: /tmp/test_jinja.txt
```

Jinja : les conditions



tests particuliers

Pareil avec number, mapping (comme un dictionnaire), iterable (liste, dict)

Ces conditions sont intéressantes dans le cas où var1 peut parfois être une valeur simple et parfois une valeur composée

```
- name: conditions avec jinja3
  hosts: all

  vars:
    var1: 1
    #var2: 2

  tasks:
    - name: copy
      copy:
        content: |
          {% if var1 is string %}
            var1 est une chaîne de caractères
          {% else %}
            var1 n'est pas une chaîne de caractères
          {% endif %}
        dest: /tmp/test_jinja.txt
```

Jinja : les conditions



tests particuliers

Pareil avec number, mapping (comme un dictionnaire), iterable (liste, dict)

Ces conditions sont intéressantes dans le cas où var1 peut parfois être une valeur simple et parfois une valeur composée

Cela permet d'éviter un plantage selon le type de variable

Rq: une chaîne de caractères est itérable

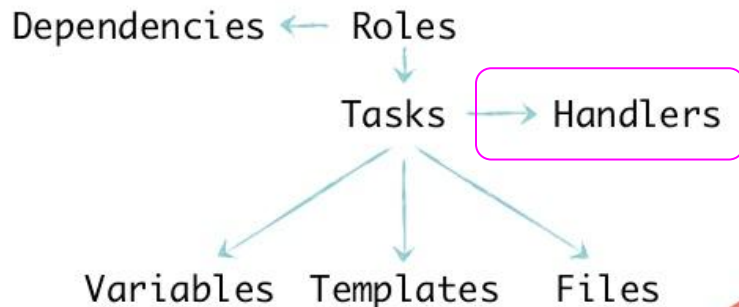
```
- name: conditions avec jinja3
hosts: all

vars:
  var1:
    - k1 : v1
    - k2 : v2
  #var2: 2

tasks:
- name: copy
  copy:
    content: |
      {% if var1 is iterable %}
      {% for k in var1 %}
      item : {{ k }}
      {% endfor %}
      {% else %}
      simple: {{ var1 }}
      {% endif %}
    dest: /tmp/test_jinja.txt
```


Ansible Handlers

ANSIBLE STRUCTURE



Ansible Handlers



Documentation: https://docs.ansible.com/ansible/latest/user_guide/playbooks_handlers.html

handlers = trigger/déclencheur

exemple : vhost nginx et reload

Ils se déclarent comme des tasks.

Principe : quand on a une action qui a réussi et qui est dans un état changed, on va pouvoir lui passer un notify qui portera le nom du handler.

Donc on déclarera une liste de handlers qui seront exécutés dès que l'action sera à changed et réussie

Ansible Handlers



Installation d'un serveur nginx sur 4 machines :

```
- name: test des handlers
  hosts: all
  become: yes
  vars:
    nginx_port: 8888
  tasks:
  - name: install nginx et curl
    apt:
      name: nginx,curl
      state: present
      cache_valid_time: 3600
      update_cache: yes
```

l'update_cache fait une mise à jour du cache apt

cache_valid_time permet de ne pas le remettre à jour pendant ici une heure

La mise à jour du cache apt sera effectuée avant l'installation des nginx et curl

Ansible Handlers



Puis nous allons supprimer chacun des vhosts qui sont installés par défaut sur les nginx, donc effacer le fichier default qui est dans **sites-available** et le lien symbolique qui est dans **sites-enabled**.

Avant cela on va récupérer la config par défaut en récupérant les lignes non commentées (qui ne contiennent pas de hashtag)

```
cat /etc/nginx/sites-available/default | grep -v "#"
```

```
sudo vim vhost.conf.j2
```



y coller la conf récupérée au-dessus

```
ansible-playbook -i 00_inventory.yml playbook.yml
```

Ansible Handlers



Maintenant nous allons supprimer nos vhosts par défaut. On ajoute à notre playbook :

```
- name: remove default file
  file:
    path: "{{ item }}"
    state: absent
  with_items:
    - "/etc/nginx/sites-available/default"
    - "/etc/nginx/sites-enabled/default"
```

Il est recommandé de mettre le format jinja entre guillemets

```
ansible-playbook -i 00_inventory.yml playbook.yml
```

Ansible Handlers

Puis nous allons utiliser une variable de groupe :

```
vim group_vars/all.yml
```

```
nginx_port: 8888
```

Ensuite nous allons ajouter au playbook notre installation grâce à notre template, là il va juste installer le fichier en question, mais le port 80 dans la conf envoyé n'a pas été templatisé

```
- name: install vhost
  template:
    src: vhost.conf.j2
    dest: /etc/nginx/sites-available/vhost.conf
    owner: root
    group: root
    mode: 0644
```

Ansible Handlers



Pour insérer le template dans le fichier vhost sur le node manager:

```
server {  
    listen {{ nginx_port }} default_server;  
    listen [::]:{{ nginx_port }} default_server;
```

Le souci c'est qu'à chaque fois que nous avons un changement du port du nginx, il nous faut un reload du nginx.

La 1^{ère} chose à faire est de créer un lien symbolique dans sites-enabled

Ansible Handlers



```
- name: création du lien symbolique du vhost
  file:
    src: /etc/nginx/sites-available/vhost.conf
    dest: /etc/nginx/sites-enabled/vhost.conf
    state: link
```

Vérification du lien symbolique sur machine distante : (couleur différente également)

```
ls /etc/nginx/sites-enabled/vhost.conf
```


Ansible Handlers



Il faut maintenant gérer le reload de nginx. Avant tout, dans tous les cas, il va falloir le démarrer une fois que tous les réglages ont été lancés.

```
- name: start nginx
  systemd:
    name: nginx
    state: started
```

Pour jouer le handler, il va juste falloir déclarer dans notre template un notify

```
- name: installation du vhost
  template:
    src: vhost.conf.j2
    dest: "/etc/nginx/sites-available/vhost.conf"
    owner: root
    group: root
    mode: 0644
    notify: reload_nginx
```

Ansible Handlers



Ensuite nous rajoutons un handler qui va gérer ces changements. Il équivaut à une task mais sous forme de trigger

```
handlers:  
- name: reload_nginx  
  systemd:  
    name: nginx  
    state: reloaded
```

Pour tester : changer la variable de groupe all (group_vars/all.yml) à 7777 par ex. On relance le playbook et on observe le lancement du handler. Puis on teste avec un `curl localhost:7777` sur la machine distante

Ansible Handlers



Les handlers s'effectuent par défaut à la fin du playbook. On peut anticiper ce comportement en faisant un flush-handlers, en lui disant à quel moment les exécuter.

Dans notre cas, l'idéal serait de relancer le reload juste après l'installation du vhost.

Si on rajoute par exemple une tâche banale comme une création de fichier juste après le start nginx et nous changeons également la variable du port nginx.

```
- name: création d'un fichier
  file:
    path: /tmp/newfile.txt
    state: touch
```

On observe, malgré la position du notify au-dessus de la création de fichier, que le handler par défaut agit toujours en dernière position

Ansible Handlers



Cependant, ATTENTION, quand on flush les handlers, ils sont tous flushés.

Donc, ici, nous rajoutons, juste sous le notify un rappel pour flush le handler

```
- name: Flush handlers  
  meta: flush_handlers
```

On re-modifie le port nginx
ON rejoue le playbook

Ansible Handlers

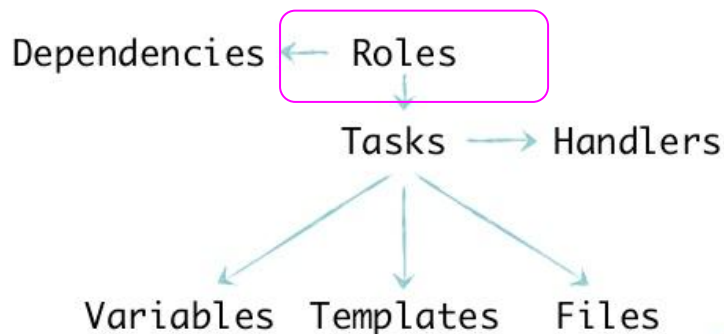


Dernière remarque sur les handlers : il est possible de coupler une condition au lancement du handler

```
- name: Check if need to restart
  stat:
    path: /var/run/reboot.pending
  register: __need_reboot
  changed_when: __need_reboot.stat.exists
  notify: reboot_server
```

Ansible Roles

ANSIBLE STRUCTURE



RÔLES : C'EST QUOI



Documentation : https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html

- C'est un répertoire qui regroupe des actions avec un objectif commun
- Il comprend également des fichiers qui lui sont liés : templates, files...
- Comme il va permettre de jouer des actions, il va aussi pouvoir appeler des variables stockées dans des fichiers

RÔLES : BONNES PRATIQUES



- Brique fondamentale de partage d'Ansible (rôles partagés sur Ansible Galaxy)
- Même si vous ne partagez pas vos rôles, le but est de pouvoir le réutiliser autant de fois que possible dans votre infra
- Notion de partage et de réutilisation : mettre en place un dépôt par rôle (on ne git pas un ensemble de rôles)
- Principe du lego : plus il est petit et plus il est réutilisable
- Au plus il est gros, au moins le rôle sera disponible dans d'autres infra (donc mieux → 1 rôle pour Prometheus, 1 rôle Grafana, 1 rôle par exporter...)

RÔLES : BONNES PRATIQUES (SUITE)



- Plus la brique sera petite, plus il sera facile d'organiser et de varier leur ordre selon les besoins
- Attention : bien gérer les dépendances et la mise en place des variables (choisir certaines par défaut par exemple pour éviter l'attente d'une variable venant d'un autre rôle)
- En résumé : inutile de créer des rôles qui ne seront utilisés qu'une seule fois
- Pas de règles sur la granularité : en fonction des besoins (possibilité de faire un rôle avec une seule task à l'intérieur)
- Un autre objectif : lorsque qqun entre dans l'équipe, en connaissant les bonnes pratiques, il pourra très rapidement être opérationnel sur votre infra

RÔLES : BONNES PRATIQUES (SUITE)



→ EX : 3 rôles pour la gestion d'une BDD

- ◆ un moteur de BDD postgres
- ◆ la réplication
- ◆ système de backup

→ Avoir des dépôts similaires pour avoir des réf communes pour fluidifier les échanges

RÔLES : ORGANISATION DE TRAVAIL



→ QUESTION : comment travailler en commun ?

- ◆ un dépôt par rôle
- ◆ qui fait quoi?
- ◆ un mainteneur du rôle (approuve les Merge Requests, Pull Requests...)
- ◆ tests des rôles : pas d'impact global trop important par rapport à la fonction déjà actuelle du rôle (ex : changement de variables de string à list, ne pas perdre des fonctionnalités...)

RÔLES : STRUCTURE



- Un rôle est une arborescence de répertoires et de fichiers yaml
- Tous ces répertoires ont pour principal fichier le main.yaml
- QUELS RÉPERTOIRES ?
 - ◆ **tasks** : concentre toutes les actions, c'est le point d'entrée du rôle, c'est le main.yml qui est lancé
 - ◆ **defaults** : les variables par défaut, quand on va tester le rôle, cela permettra d'éviter les messages d'erreur et aussi de setter des variables qui évoluent peu, ou pour définir un comportement par défaut du rôle. Ces variables ont vocation à être surchargées. Ce sont celles qui sont le moins pris en compte
 - ◆ **vars** : les variables de rôle (variables structurées du rôle importantes qui évolueront peu)

RÔLES : STRUCTURE



→ QUELS RÉPERTOIRES ?

- ◆ **handlers** : les déclencheurs (sur un notify, changed par exemple, on va appeler un handler qui va déclencher une/des task(s))
- ◆ **templates** : fichiers templates au format jinja (mettre une extension .j2)
- ◆ **files** : les fichiers à copier ou fichiers statiques (éviter au max les binaires)
- ◆ **meta** : pour partager sur galaxy et inclure des dépendances (charger des rôles dont dépend le rôle en action)
- ◆ **test** : éléments de tests (ex avec Molecule), voir “universalité” du rôle
- ◆ **library** : utilisation de modules développés par nos soins

RÔLES : GALAXY



- <https://galaxy.ansible.com/>
- On y retrouve des rôles et également maintenant des collections
- Développés par les propriétaires de l'outil ou par d'autres notamment geerlingguy
- Ansible Galaxy ne stocke pas les rôles, ils sont déposés sur GitHub
- Lorsque vous allez sur le repo d'un rôle, vous observez toujours la même structure
- Ligne de commande ansible-galaxy

RÔLES : GALAXY



Créer un répertoire **roles** juste à côté du playbook

```
mkdir roles
```

```
cd roles
```

```
ansible-galaxy init monrole
```

```
tree
```

```
cd ..
```

```
tree
```

Rq : pour GitHub, il faut seulement le faire avec le dossier monrole et non depuis le dossier racine avec le playbook et l'inventary

RÔLES : GALAXY



Pour GitHub, il faut seulement le faire avec le dossier monrole et non depuis le dossier racine avec le playbook et l'inventary

```
git init (dans monrole)
```

```
vim .gitignore (dans racine)
```



```
./roles/*
```


RÔLES : MISE EN PRATIQUE



3 rôles : (utilisation de VSC plus pratique)

- **ssh_keygen** : génération de la clef en local (pb become yes)
- **users** : création des users et déploiement des clefs
- **nginx** : installation d'un reverse proxy

1 MANAGER ET 4 NODES (mettre à jour le fichier 00_inventory.yml)

```
mkdir roles
```

```
ansible-galaxy init roles/ssh_keygen  
ansible-galaxy init roles/users  
ansible-galaxy init roles/nginx
```

RÔLES : MISE EN PRATIQUE



Modification du playbook en 2 zones :

- installation en local de la clef ssh
- installation des serveurs (avec users et nginx)

Passage de rôles à la place des tasks

On prépare ainsi avant l'appel des rôles

```
- name: Installation locale de la clef ssh
  connection: local
  hosts: localhost
  roles:
    - ssh_keygen

- name: installation des serveurs (users,
  nginx)
  hosts: all
  become: true
  roles:
    - users
    - nginx
```

RÔLES : MISE EN PRATIQUE



Ajout de la tâche dans main.yml dans le répertoire tasks du rôle ssh_keygen

```
roles/ssh_keygen/tasks/main.yml
```

```
- name: generate SSH key
  openssh_keypair:
    path: /tmp/key_hosts
    type: rsa
    size: 4096
    state: present
    force: false
```

Avec le module openssh_keypair
path ici en local
on ne force pas sa régénération

Même si ici l'exemple est simple, le but
est de diviser au maximum le travail
pour y voir le plus clairement possible

RÔLES : MISE EN PRATIQUE



Création du second rôle avec la création du user et ajout de la clé ssh **sur les machines distantes**

```
roles/users/tasks/main.yml
```

Rq : pas besoin du become: yes pour ces tâches car déjà cité de façon globale dans le playbook.yml

```
- name: création du user devops
  user:
    name: devops
    shell: /bin/bash
    groups: sudo
    append: yes
    password: "{{ 'password' | password_hash('sha512') }}"

- name: Add devops user to the sudoers
  copy:
    dest: "/etc/sudoers.d/devops"
    content: "devops ALL=(ALL) NOPASSWD: ALL"

- name: Deploy SSH Key
  authorized_key:
    user: devops
    key: "{{ lookup('file', '/tmp/key_hosts.pub') }}"
    state: present
```

RÔLES : MISE EN PRATIQUE



Création du 3e rôle avec la création du serveur nginx avec l'utilisation d'un template

Le template va nous permettre de reproduire du code, ici la conf de nginx

Récupération du code pour un virtual host dans

```
cat /etc/nginx/sites-available/default | grep -v "#"
```

RÔLES : MISE EN PRATIQUE



Création du 3e rôle avec la création du serveur nginx avec l'utilisation d'un template

Le template va nous permettre de reproduire du code, ici la conf de nginx

```
roles/nginx/templates/default_vhost.conf.j2
```

Bonne pratique : préfixer toutes les variables en fonction du nom du rôle

RÔLES : MISE EN PRATIQUE



Création du 3e rôle avec la création du serveur nginx avec l'utilisation d'un template

```
server {  
    listen {{ nginx_port }} default_server;  
    listen [::]:{{ nginx_port }} default_server;  
    root /var/www/html;  
  
    index index.html index.htm index.nginx-debian.html;  
  
    server_name _;  
  
    location / {  
        try_files $uri $uri/ =404;  
    }  
}
```


RÔLES : MISE EN PRATIQUE



Création du 3e rôle avec la création du serveur nginx avec l'utilisation d'un template

Maintenant on va s'intéresser aux tasks, on ouvre le main.yml des tasks

On installe nginx avec le module apt

On supprime la config par défaut de nginx avec with_items

On installe notre template vhost qui va directement chercher le fichier dans le template du rôle et va créer le fichier default_vhost en local

Puis création d'un lien symbolique (state: link) pour utiliser la conf que nous avons envoyée

On démarre ensuite le nginx surtout avant de l'appeler avec le flush_handlers qui est censé faire un reload. Il plantera s'il fait un reload sur un nginx non démarré.

```

- name: install nginx
  apt:
    name: nginx,curl
    state: present
    cache_valid_time: 3600
    update_cache: yes

- name: remove default file
  file:
    path: "{{ item }}"
    state: absent
  with_items:
    - "/etc/nginx/sites-available/default"
    - "/etc/nginx/sites-enabled/default"

- name: install vhost
  template:
    src: default_vhost.conf.j2
    dest: /etc/nginx/sites-available/default_vhost.conf
    owner: root
    group: root
    mode: 0644
    notify: reload_nginx

- name : création d'un lien symbolique
  file:
    src: /etc/nginx/sites-available/default_vhost.conf
    dest: /etc/nginx/sites-enabled/default_vhost.conf
    state: link

- name: start nginx
  service:
    name: nginx
    state: started

- name: Flush handlers
  meta: flush_handlers

```

Rôle nginx :

```

- name: start nginx
  systemd:
    name: nginx
    state: started

```

Puis nous allons définir le main.yml du handlers

```

- name: reload_nginx
  service:
    name: nginx
    state: restarted

```

RÔLES : MISE EN PRATIQUE



Puis nous pouvons indiquer le port par défaut dans le main.yml de defaults de nginx

```
nginx_port: 80
```

```
ansible-playbook -i 00_inventory.yml playbook.yml
```

Maintenant nous pouvons nous connecter avec le user devops en ssh avec la clé qui est dans le /tmp de notre localhost

```
ssh -i /tmp/devops devops@192.168.15.11
```

RÔLES : MISE EN PRATIQUE



Test ensuite du serveur nginx sur la machine distante

```
curl 127.0.0.1
```

Mais en revanche nous n'avons pas de connexion sur 127.0.0.1:8888

Nous allons pouvoir utiliser les group_vars

Création de all.yml dans le dossier group_vars → `nginx_port: 8888`

Cela va nous permettre de surcharger le port

RÔLES : MISE EN PRATIQUE



Mais nous aimerions choisir un autre port pour notre nginx sur un node en particulier

Donc création d'un fichier yaml dans le host_vars qui va se nommer avec l'IP de l'hôte

192.168.15.13.yml

```
nginx_port: 7777
```

Exploiter des variables au format json



1. Créez un fichier JSON nommé variables.json

```
{  
  "nginx_version": "latest",  
  "app_port": "8080",  
  "container_name": "myapp",  
  "compose_project_name": "myproject",  
  "compose_file_path": "/path/to/docker-compose.yml"  
}
```

Exploiter des variables au format json



2. Créez un fichier docker-compose.yml dans le même répertoire que votre fichier JSON

```
version: '3'
services:
  web:
    image: nginx:{{ nginx_version }}
    ports:
      - "{{ app_port }}:80"
    container_name: "{{ container_name }}"
```

Exploiter des variables au format json

3. Créez un playbook Ansible nommé `deploy.yml` pour charger les variables depuis le fichier JSON et exécuter Docker Compose.

```
---
- name: Déployer des conteneurs Docker avec Docker Compose
  hosts: all
  tasks:
    - name: Charger les variables depuis un fichier JSON
      include_vars:
        file: variables.json

    - name: Lancer Docker Compose
      command: docker-compose up -d
      args:
        chdir: "{{ compose_file_path }}"
```



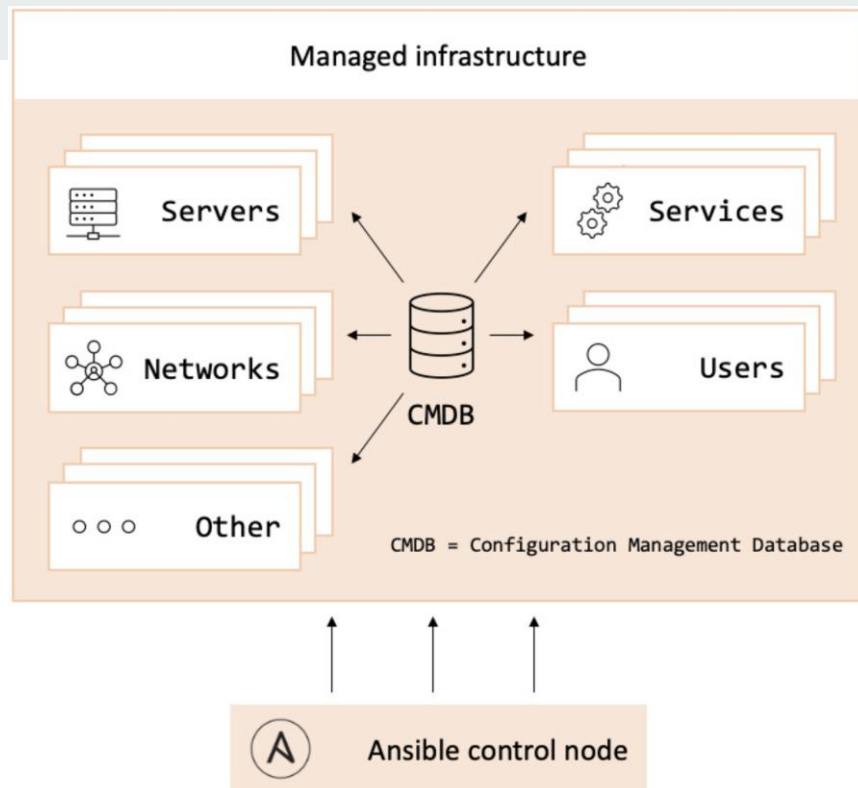
`ansible-playbook deploy.yml`

CMDB

Les environnements d'entreprise de production incluent généralement une base de données de gestion de configuration (CMDB) pour organiser leurs actifs d'infrastructure informatique.

Des exemples d'actifs d'infrastructure informatique sont les serveurs, les réseaux, les services et les utilisateurs.

Bien qu'elle ne fasse pas directement partie de l'architecture Ansible, la CMDB décrit les actifs et leurs relations au sein d'une infrastructure gérée et peut être exploitée pour créer un inventaire Ansible.



L'inventaire est soit déduit de la CMDB, soit créé manuellement par l'administrateur système.

CMDB



<https://ansible-cmdb.readthedocs.io/en/latest/>

<https://www.hashicorp.com/resources/how-to-deploy-your-hashicorp-stack-with-ansible-in-under-15-minutes>

<https://blog.wescale.fr/ansibled-consul-securise>

Host Overview

Generated on Sun Nov 29 11:10:33 2015

Shown columns

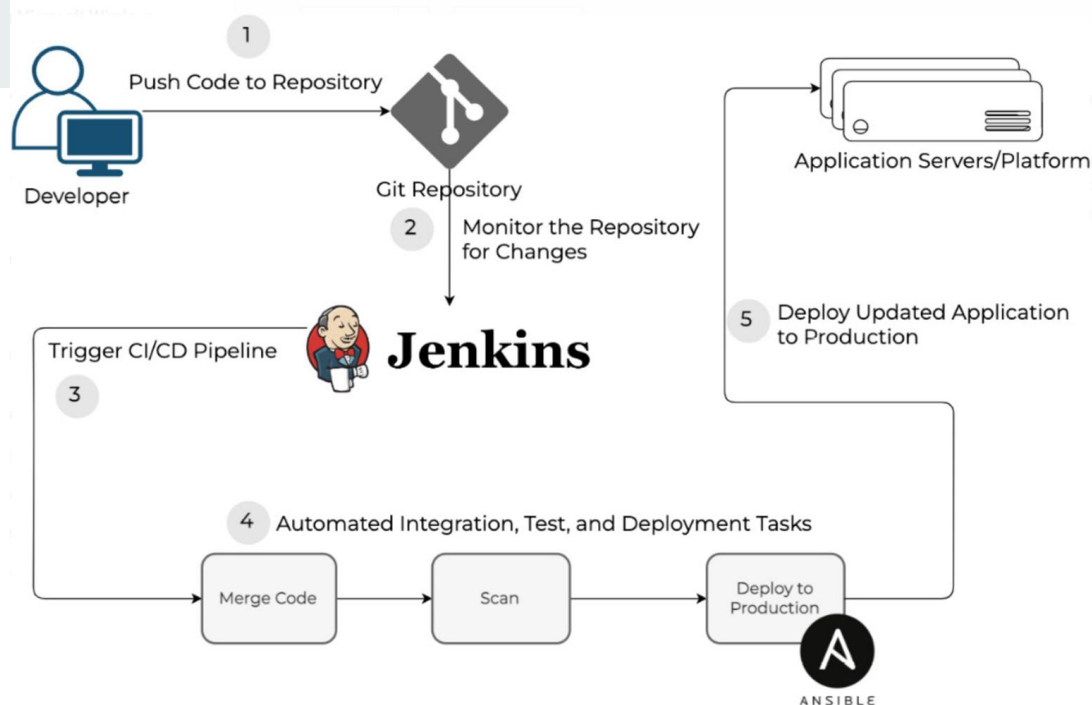
Name	Groups	DTAP	Comment	Ext ID	FQDN	Main IP	All IPv4	OS
Kernel	Arch	Virt	CPU type	vCPUs	RAM [GiB]	Mem Usage	Swap Usage	
Disk usage	Timestamp							

Host overview

Name	DTAP	Main IP	OS	vCPUs	Mem Usage
app.uat.local	uat	192.168.57.1	Debian 6.0.10	1	<div><div></div></div> (0.1 / 0.5 GiB)
centos.dev.local	dev	192.168.56.8	CentOS 6.6	1	<div><div></div></div> (0.1 / 1.0 GiB)
db01.prod.local	prod	192.168.58.1	Debian 6.0.10	1	<div><div></div></div> (0.1 / 0.5 GiB)
db02.prod.local	prod	192.168.58.2	Debian 6.0.10	1	<div><div></div></div> (0.1 / 0.5 GiB)
db03.prod.local	prod	192.168.58.3	Debian 6.0.10	1	<div><div></div></div> (0.1 / 0.5 GiB)
debian.dev.local	dev	192.168.56.2	Debian 6.0.10	1	<div><div></div></div> (0.1 / 0.5 GiB)
eeek.electricmonk.nl	prod	192.168.0.10	Ubuntu 14.04	2	<div><div></div></div> (0.9 / 2.9 GiB)

Generated by ansible-cmdb v%%MASTER%% - © Ferry Boender 2015

DevOps : insérer Ansible



Ansible peut servir à beaucoup d'autres tâches :

- Scanner l'application
- Création d'artefacts d'application
- Exécution de tests unitaires et d'intégration
- Promouvoir et tester l'application dans l'environnement de développement
- Stockage des artefacts d'application dans le référentiel d'artefacts
- Déploiement de l'application en production



Précédence des Variables

Précédence des Variables



Nous revoyons rapidement les 23 types de variables

```
command line values (eg "-u user")
role defaults [1]
inventory file or script group vars [2]
inventory group_vars/all [3]
playbook group_vars/all [3]
inventory group_vars/* [3]
playbook group_vars/* [3]
inventory file or script host vars [2]
inventory host_vars/* [3]
playbook host_vars/* [3]
host facts / cached set_facts [4]
play vars
play vars_prompt
play vars_files
role vars (defined in role/vars/main.yml)
block vars (only for tasks in block)
task vars (only for the task)
include_vars
set_facts / registered vars
role (and include_role) params
include params
extra vars (always win precedence) (-e)
```

Précédence des Variables



playbook.yml

```
- name: test precedence variables
hosts: all
vars:
  var1: "playbook variables"
roles:
- test_variables
```

```
mkdir roles
```

```
ansible-galaxy init roles/test_variables
```

Précédence des Variables



En premier, il y a la variable par défaut de chaque rôle

Dans roles/tasks/main.yml

```
- name: debug
  debug:
    var: var1
```

```
ansible-playbook -i 00_inventory.yml playbook.yml
```

Affiche que la variable n'est pas définie parce que nous sommes dans un debug, mais planterai si pas dans un debug

Précédence des Variables



Donc pour définir au minimum cette variable est dans le defaults/main.yml

```
var1: "default role"
```

```
ansible-playbook -i 00_inventory.yml playbook.yml
```

Maintenant on veut surcharger cette variable variables de groupes dans group_vars/all.yml

```
var1: "group vars"
```

```
ansible-playbook -i 00_inventory.yml playbook.yml
```


Précédence des Variables



Mais au sein d'un groupe j'aimerais donner cette variable directement à une machine dans `host_vars/<IP_machine>.yaml`

```
var1: "host vars"
```

```
ansible-playbook -i 00_inventory.yaml playbook.yaml
```

Surcharge aussi dans le playbook :

```
- name: test variables
  hosts: all
  vars:
    var1: "playbook variables"
  roles:
    - test_variables
```

```
ansible-playbook -i 00_inventory.yaml playbook.yaml
```

Précédence des Variables



Puis celle qui s'impose le plus au niveau du rôle sera dans vars/main.yml

```
var1: "vars roles"
```

```
ansible-playbook -i 00_inventory.yml playbook.yml
```

Puis on peut aussi enregistrer une variable dans les gather_facts dans tasks/main.yml :

```
- name: set fact
  set_fact:
    var1: "fact roles"

- name: debug
  debug:
    var: var1
```

Précédence des Variables



Puis dans la ligne de commande :

```
ansible-playbook -i 00_inventory.yml -e "var1='cli var'" playbook.yml
```

Ansible Collections



Ansible Collections



Un petit aperçu

Quels problèmes sont résolus avec les collections ?

Comment en tant que développeur de playbook vais-je utiliser ce nouveau concept ?

Ansible Collections



Quand vous visitez le dépôt Ansible <https://github.com/ansible/ansible> , vous vous rendez vite compte de deux choses :

1. grand nombres de contributeurs (plus de 5000)
2. Énormément d'issues à gérer

Solution :

- Basculer les contributions de la communauté hors du projet principal pour que les modules ou autres plugins puissent avoir un cycle de release plus court et indépendant du coeur d'Ansible
- Utilisation d'Ansible Galaxy
- Nouveau concept de collections depuis la version 2.9

Ansible Collections

→ Qu'est-ce qu'une COLLECTION ?

C'est une archive qui est structurée de cette façon

Les rôles sont bien entendu gardés et ceux qui en ont déjà créé pourront les y placer

Mais on peut y ajouter des modules, des filtres....

```
plugins/modules : pour vos modules.  
plugins/inventory : pour vos plugins d'inventaire.  
plugins/filter : pour vos filtres.  
plugins/module_utils : pour le code commun à vos plugins
```

```
collection/  
├─ docs/  
├─ galaxy.yml  
├─ plugins/  
│   ├─ modules/  
│   │   └─ module1.py  
│   ├─ inventory/  
│   │   └─ .../  
├─ README.md  
├─ roles/  
│   ├─ role1/  
│   ├─ role2/  
│   └─ .../  
└─ playbooks/
```

Ansible Collections



Pour créer une collection, créer un répertoire ansible-collections (le nom est important)

```
ansible-galaxy collection --help
```

La commande init nous permet de créer l'arborescence de base d'une collection :

```
ansible-galaxy collection init essai.my_collection
```

```
tree
```


Ansible Collections



Créons maintenant un rôle debug qui affichera simplement un message de debug

```
ansible-galaxy role init debug
```

Vous pouvez mettre les bonnes informations dans `essai/my_collection/roles/debug/meta/main.yml`

```
# essai/my_collection/roles/debug/tasks/main.yml
---

- name: mon debug
  debug:
    msg: "{{ ansible_default_ipv4.address }}"
```

Ansible Collections



Créons un filtre jinja3 juste pour voir comment un filtre est utilisé :

```
# essai/my_collection/plugins/filter/my_custom_filters.py

from __future__ import (absolute_import, division,
print_function)
__metaclass__ = type

def split_ip(value):
    return value.split(".")

class FilterModule(object):

    def filters(self):
        return {
            'split_ip': split_ip
        }
```

Renvoie une liste avec les
IP

Ansible Collections

Retournons au rôle et utilisons ce filtre dans une task :

```
# essai/my_collection/roles/debug/tasks/main.yml
---
- name: mon debug
  debug:
    msg: "{{ ansible_default_ipv4.address | essai.my_collection.split_ip }}"
...
```

Ansible Collections



Afin de pouvoir utiliser notre collection nous devons préciser à Ansible son emplacement car nous n'avons pas créé celle-ci dans l'un des emplacements reconnu par défaut :

- `$HOME/.ansible/collections`
- `/usr/share/ansible/collections`

Note : Vous remarquerez que ces emplacements par défaut ne contiennent pas le fameux répertoire `ansible_collections`

Nous pouvons utiliser deux méthodes pour ajouter un chemin vers des collections :

A travers `ansible.cfg` en ajoutant le paramètre `collections_paths` dans la section `defaults`

A travers la variable d'environnement `ANSIBLE_COLLECTIONS_PATHS`

Les 2 méthodes nécessitent de renseigner la liste complètes des emplacements de collections. Nous allons donc créer le fichier `ansible.cfg` dans notre répertoire avec le contenu suivant:

Ansible Collections



```
# ansible.cfg
[defaults]
collections_paths = ~/.ansible/collections:/usr/share/ansible/collections:<ansible collections parent directory>
```

Créer ensuite le playbook qui va utiliser notre collection

```
# test.yml

- hosts: localhost
  gather_facts: true
  tasks:
    - import_role:
      name: essai.my_collection.debug
```

```
ansible-playbook test.yml
```



Pour s'entraîner

https://ansible.github.io/workshops/exercises/ansible_network/