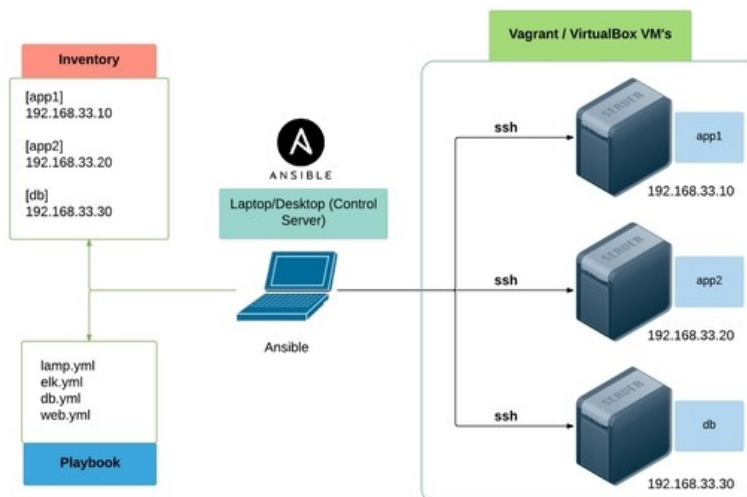


Ansible

Présentation

Ansible est une plateforme logicielle libre d'**Infrastructure As Code** (IaC) pour l'automation de la configuration et de la gestion des ordinateurs. Il gère les différents nœuds à travers **SSH** et ne nécessite l'installation d'aucun logiciel supplémentaire sur ceux-ci (agentless). Ce qui simplifie son utilisation, sa maintenance, sa mise à l'échelle, et n'utilise pas de ressources supplémentaires sur les machines à gérer. C'est un outil idempotent, ce qui signifie qu'une opération a le même effet qu'on l'applique une ou plusieurs fois, par exemple, plutôt qu'ajouter une ligne à fichier, on s'assurera que la ligne en question est bien présente dans le dit fichier. Et enfin, **Ansible** utilise une approche impérative qui définit comment atteindre le but souhaité (comment faire ?) par opposition à un langage déclaratif comme **Terraform** qui décrit l'état attendu (quoi faire ?).



Installation

Sur la machine depuis laquelle on va lancer nos commandes ansible :

Debian : `sudo apt install ansible`

RedHat : `sudo dnf install ansible-core`

L'utilisateur ansible

Sur les machines à administrer :

Il est conseillé, sur chaque machine à administrer, de créer un utilisateur dédié, que nous appellerons ici **ansible...**

Il faudra ensuite donner tous les droits sudo sans mot de passe à cet utilisateur en créant un fichier /etc/sudoers.d/ansible :

```
ansible ALL=(ALL) NOPASSWD:ALL
```

Sur le contrôleur :

Ne pas utiliser sudo !!!

On génère un fichier ansible.cfg de base pour l'utilisateur en charge d'**ansible**

```
ansible-config init --disabled > ~/.ansible.cfg
```

On lui génère une paire de clefs SSH

```
ssh-keygen
```

On copie sa clef publique sur chaque machine à administrer :

```
ssh-copy-id ansible@host
```

On peut ensuite sur chaque machine locker le password de l'utilisateur ansible qui ne pourra plus se connecter qu'avec sa clef :

```
sudo passwd -d ansible
```

```
sudo passwd -l ansible
```

L'inventaire :

On doit d'abord générer un fichier de préférences pour ansible :

```
ansible-config init --disabled > ~/.ansible.cfg
```

Ce qui crée en même temps un répertoire caché ~/.ansible

On va maintenant pouvoir écrire notre inventaire :

```
vi ~/.ansible/hosts
```

Exemple de fichier hosts :

```
server ansible_host=ip_server      ansible_user=ansible ansible_become=true
```

```
poste1 ansible_host=ip_poste1      ansible_user=root
```

```
poste2 ansible_host=ip_poste2      ansible_user=root
```

```
[local]
```

```
poste1
```

poste2

Ici, on a un utilisateur ansible sur server qui exécute les commandes en sudo (ansible_become) alors que sur les poste1 et 2, c'est l'utilisateur root qui exécute les commandes !

On a ensuite créé un groupe de machines « local ».

On peut aussi utiliser :

ansible_port si on utilise un autre port que le port 22 pour SSH

ansible_ssh_pass si vous utilisez un mot de passe pour SSH

ansible_ssh_private_key_file si vous devez utiliser plusieurs clés spécifiques aux hôtes

On doit maintenant modifier notre fichier de préférences ansible.cfg :

```
inventory=~/.ansible/hosts
```

On peut examiner notre inventaire avec la commande :

```
ansible-inventory --list
```

```
ansible-inventory --graph
```

```
ansible-inventory --host poste1
```

N.B. On peut aussi indiquer le fichier inventaire en ligne de commande :

```
ansible-inventory -i fichier_inventaire --graph
```

Les modules

Un module Ansible est un script autonome réutilisable qui effectue une tâche ou une opération spécifique sur un hôte distant. Les modules peuvent gérer les ressources du système comme les paquets, les services, les fichiers et les utilisateurs, entre autres choses.

ansible-doc

Pour trouver la documentation sur un module

```
ansible-doc nom_module
```

Ou le site : <https://docs.ansible.com>

Patterns and ad-hoc commands

Les commandes ad-hoc sont des tâches ponctuelles en ligne de commande qui permettent d'exécuter des opérations simples sur des hôtes distants. Ils sont utiles pour des tâches rapides telles que la

gestion de fichiers, le redémarrage de serveurs ou la vérification de la connectivité.

ansible (-i fichier_inventaire -u remote_user) -m module -a module_arguments host|group|all (--options)

Les options utiles :

--verbose (-v)

--check (-C)

--diff (-D) # utile en combinaison avec check

N.B. En l'absence de clef ssh, pour des tests, il nous faudra installer (le cas échéant) le programme sshpass et utiliser les options suivantes :

--user : utilisateur distant

--ask-pass : pour entrer le mot de passe ssh

--become : pour se connecter en sudo

--ask-become-pass : pour entrer le mot de passe sudo

Premiers modules

Le module command :

ansible -m command -a uptime all

Le module command est le module par défaut, on peut donc simplifier la commande ad-hoc précédente en :

ansible -a uptime all

Ou encore :

ansible -a "echo Hello World" localhost

N.B. Localhost n'a pas besoin d'être dans l'inventaire !

Attention :

ansible -a "free -h" all

mais :

ansible -a "ps -ef|wc -l" all nous renvoie une erreur :

Les commandes ne sont pas traitées par le shell, de sorte que les opérations telles que « * », « », « | », « ; » et « & » ne fonctionnent pas. De plus, les variables d'environnement sont résolues par

Python et non par le shell. Nous devons donc utiliser le module **shell** si nous avons besoin de ces fonctionnalités :

```
ansible -m shell -a "ps -ef|wc -l" all
```

Il existe aussi le module **raw** qui ne nécessitant pas python (indispensable à **ansible**) va permettre de l'installer sur les machines sur lesquelles il n'est pas installé (dans le cas d'un container Debian par exemple):

```
ansible -m raw -a "apt install -y python3" debian
```

Il existe enfin le module script pour exécuter un script local :

```
ansible -m script -a "/home/$USER/hello.sh" all
```

N.B. À ce stade, les commandes ad-hoc command, shell, raw et script ne sont pas idempotentes !

Le module ping :

```
ansible -m ping all
```

Un module de test trivial, qui retourne toujours 'pong' sur un contact réussi. Cela n'a pas de sens dans un playbook (voir ci-dessous), mais sa commande ad-hoc est utile pour vérifier la capacité de login et la présence d'un Python utilisable.

N.B. En cas d'avertissement sur l'interpréteur python, on peut modifier le fichier ansible.cfg :

```
interpreter_python=auto_silent
```

Pour travailler sans fichier inventaire :

```
ansible -i ip_poste1,ip_poste2 -u root -m ping all
```

N.B. Si un seul host, rajouter une virgule à la fin :

```
ansible -i ip_poste1, -u root -m ping all
```

Ansible facts et le module setup :

Les variables liées aux systèmes distants sont appelées facts.

```
ansible -m setup all (récolte les ansible_facts)
```

Pour filtrer sur une variable :

```
ansible -m setup -a "filter=ansible_distribution" all
```

```
ansible -m setup -a "filter=ansible_os_family" all
```

```
ansible -m setup -a "filter=ansible_hostname" all
```

Les playbooks

Les **playbooks Ansible** sont des modèle de tâches d'automatisation écrits en **YAML**, qui indiquent à **Ansible** les actions à réaliser et sur quelles machines.

*N.B. **YAML** : **Y**AML **A**in't a **M**arkup **L**anguage*

```
- name : mon playbook
```

```
  hosts : host|group|all
```

```
  tasks:
```

```
    - name : ma première tâche
```

```
      module : module ansible
```

```
        arg1
```

```
      ...
```

```
    - name : ma seconde tâche
```

```
      module :
```

```
        ...
```

N.B. Dans les playbooks suivant de ce doc, c'est vrai que les plays (ensemble des tâches) ne sont pas nommés. Ce sera corrigé dans une prochaine version...

Allons-y :

```
mkdir playbooks
```

Un simple playbook :

```
vi playbooks/uptime.yml
```

```
- hosts: all
```

```
  tasks:
```

```
    - name: uptime
```

```
      command: uptime
```

Pour le lancer :

```
ansible-playbook playbooks/uptime.yml
```

N.B. On a les mêmes options `-i inventory -u remote_user...` et `--verbose`, `--check` et `--diff` que pour la commande `ansible...`

N.B. On peut aussi limiter l'exécution du playbook à certains hosts :

```
ansible-playbook playbooks/uptime.yml --limit poste1
```

N.B. Avant d'effectuer les tâches définies par l'utilisateur, on voit que, par défaut, le playbook récolte les différentes variables ansible (`gather_facts`) comme lors de l'utilisation du module `setup`. On verra leur utilisation plus tard. Mais si on n'en a pas besoin, on peut le préciser :

```
---
```

```
- hosts: all
```

```
  gather_facts: no
```

```
  tasks:
```

```
    - name: uptime des serveurs
```

```
      command: uptime
```

Par contre le playbook, contrairement à une commande `ad-hoc` ne nous renvoie pas la sortie de la commande mais seulement l'état des différentes tâches (dont on verra plus précisément les différentes significations plus tard) :

```
ok=1  changed=1  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```

Pour enregistrer la sortie de la commande et l'afficher :

```
- hosts: debian
```

```
  tasks:
```

```
    - name: uptime des serveurs
```

```
      command: uptime
```

```
      register: uptime_result
```

```
    - name: debug
```

```
      debug:
```

```
        #var: uptime_result
```

#on peut affiner...

var: uptime_result.stdout

On peut utiliser des variables : Ansible utilise des accolades doubles (la syntaxe **Jinja2**) pour les référencer. Par exemple :

- hosts: all

tasks:

- name: uptime des serveurs

command: uptime

register: uptime_result

- name: debug

debug:

msg: "uptime de {{ansible_hostname}} : {{uptime_result.stdout}}"

N.B. Ne pas oublier de récolter les ansible_facts !

T.P. Écrire un playbook qui teste la connectivité des serveurs, les reboot et affiche leur uptime au redémarrage.

Module avec plusieurs arguments :

copy.yml

- hosts: all

tasks:

- name: copie d'un fichier

copy:

src: /etc/hosts

dest: /tmp

Ou alors :

```
mkdir playbooks/files
```

```
cp /etc/hosts playbooks/files
```

```
---
```

```
- hosts: all
```

```
tasks:
```

```
- name: copie d'un fichier
```

```
  copy:
```

```
    src: hosts
```

```
    dest: /tmp
```

Avec des variables :

```
---
```

```
- hosts: all
```

```
vars:
```

```
  file: "/etc/hosts"
```

```
  dir: "/tmp"
```

```
tasks:
```

```
- name: copie d'un fichier
```

```
  copy:
```

```
    src: '{{ file }}'
```

```
    dest: '{{ dir }}'
```

N.B. Notez les guillemets obligatoires autour des variables lorsqu'elles débutent une expression !

```
- {{ foo }}
```

doit s'écrire :

```
- '{{ foo }}'
```

Les variables peuvent être surchargées par des extra variables en ligne de commande :

```
ansible-playbook playbooks/copy.yml -e "file=/etc/hostname" -e "dir=/root"
```

T.P. Avec le module fetch, recopier les fichiers /etc/hosts des serveurs sur votre contrôleur ansible dans /tmp !

Les conditions : when

mise à jour de systèmes Linux

- hosts: all

tasks:

- name: upgrade debian family

apt:

update_cache: yes

upgrade: full

when: ansible_os_family == 'Debian'

- name: upgrade redhat family

yum:

name: '*'

state: latest

when: ansible_os_family == 'RedHat'

N.B. Il existe aussi un module package...

- hosts: all

tasks:

- name: upgrade linux

package:

name: '*'

state: latest

État des tâches

Changed task : cf. idempotence...

N.B. reprendre uptime.yml :

```
- name: uptime des serveurs
  command: uptime
  register: uptime_result
  changed_when : false
```

Failed task :

umask.yml

```
- hosts: all

tasks:

  - name: umask

    replace:

      path: /etc/skel/.profile

      regexp: '^#*umask [0-9]*'

      replace: 'umask 002'
```

Seules les distributions de la famille debian ont des fichiers .profile (.bash_profile pour la famille RedHat))

Ignored task :

```
- hosts: all

tasks:

  - name: umask

    replace:

      path: /etc/skel/.profile

      regexp: '^#*umask [0-9]*'

      replace: 'umask 002'

  ignore_errors : true
```

Permet de passer à la tâche suivante et de finir le playbook

Skipped task :

```
- hosts: all

tasks:

  - name: umask

    replace:

      path: /etc/skel/.profile

      regexp: '^#*umask [0-9]*'

      replace: 'umask 002'

    when: ansible_os_family == 'Debian'
```

Ici, les machines redhat ont été évitées...

Rescued task :

Les blocs créent des groupes logiques de tâches. Les blocs offrent également des moyens de gérer les erreurs, de la même manière que la gestion des exceptions dans de nombreux langages de programmation.

```
- hosts: all

tasks:

  - block:

    - name: umask

      replace:

        path: /etc/skel/.profile

        regexp: '^#*umask [0-9]*'

        replace: 'umask 002'

    - name: confirmation

      debug:

        msg: "Umask configuré sur {{ansible_hostname}}"

  rescue:

    - name: affichage d'un message d'erreur

      debug:
```

```
msg: "Les distributions {{ ansible_os_family }} n'ont pas de .profile"
- name: umask redhat
  lineinfile:
    path: /etc/skel/.bash_profile
    line: umask 002
```

Un playbook pour LAMP

On va rajouter un groupe web_servers dans /etc/ansible/hosts avec des machines sous debian et rockylinux:

```
[web_servers]
```

```
...
```

```
...
```

playbooks/lamp.yml :

```
---
```

```
- hosts: web_servers
```

```
tasks:
```

```
- name: LAMP installation Debian/Ubuntu
```

```
  package:
```

```
    - apache2
```

```
    - mariadb-server
```

```
    - php
```

```
    - php-mysql
```

```
  state: present
```

```
  when: ansible_os_family == 'Debian'
```

```
- name: LAMP installation Rockylinux/Redhat
```

```
  package:
```

```
    - httpd
```

```
    - mariadb-server
```

```
    - php
```

```
- php-mysqldb  
state: present  
when: ansible_os_family == 'RedHat'
```

Avec des variables :

```
- hosts: webservers
```

```
vars:
```

```
debian_lamp_packages:
```

```
- apache2  
- mariadb-server  
- php  
- php-mysql
```

```
redhat_lamp_packages:
```

```
- httpd  
- mariadb-server  
- php  
- php-mysqldb
```

```
tasks:
```

```
- name: LAMP installation Debian/Ubuntu
```

```
package:
```

```
name : '{{ debian_lamp_packages }}'
```

```
state : present
```

```
when: ansible_os_family == 'Debian'
```

```
- name: LAMP installation Rockylinux/Redhat
```

```
package:
```

```
name : '{{ redhat_lamp_packages }}'
```

```
state : present
```

```
when: ansible_os_family == 'RedHat'
```

Par contre, si on veut se connecter sur le site web, on s'aperçoit que Redhat ne démarre pas les services à l'installation !

Les boucles : with_items loop

On doit donc rajouter une tâche qui va démarrer en boucle les différents services.

- name: activation des services sur Redhat

service:

state: started

enabled : yes

name: '{{ item }}'

with_items:

- httpd

- mariadb

when: ansible_os_family == 'RedHat'

N.B. Avec les dernières versions de ansible, on peut remplacer :

with_items :

par

loop :

Les rôles :

cd playbooks/

ansible-galaxy init lamp

tree lamp/

playbooks/lamp/

|— defaults

| └─ main.yml

|— files

|— handlers

| └─ main.yml

```
|— meta
|   └─ main.yml
|— README.md
|— tasks
|   └─ main.yml
|— templates
|— tests
|   └─ inventory
|   └─ test.yml
└─ vars
    └─ main.yml
```

N.B. Le répertoire test contient un playbook et un inventaire pour des tests automatisés. Nous ne nous en servons pas ici. Le répertoire defaults est censé séparer, par convention, les variables par défaut (constantes...) des autres variables...

lamp/vars/main.yml

redhat_lamp_packages :

- httpd
- mariadb-server
- php
- php-mysqld

debian_lamp_packages:

- apache2
- mariadb-server
- php
- php-mysql

lamp/tasks/main.yml :

- name: LAMP installation sur debian

```
package:
  name : '{{ debian_lamp_packages }}'
  state : present
when: ansible_os_family == 'Debian'
```

- name: LAMP installation sur redhat

```
package:
  name : '{{ redhat_lamp_packages }}'
  state : present
when: ansible_os_family == 'RedHat'
```

- name: activation des services sur Redhat

```
service:
  state: started
  name: '{{ item }}'
with_items:
  - httpd
  - mariadb
when: ansible_os_family == 'RedHat'
```

playbooks/lamp.yml :

```
- hosts: web_servers
roles:
  - lamp
```

Allez,décomposons encore un peu, avec des includes...

lamp/vars/main.yml

```
# variables déplacées dans RedHat.yml et Debian.yml
# voir tasks/main.yml
```

lamp/vars/RedHat.yml

lamp_packages:

- httpd
- mariadb-server
- php
- php-mysqlnd

lamp/vars/Debian.yml

lamp_packages:

- apache2
- mariadb-server
- php
- php-mysql

lamp/tasks/main.yml

- name: configuration des variables

include_vars: '{{ ansible_os_family }}.yml'

- name: LAMP installation

package:

name : '{{ lamp_packages }}'

state : present

- name: activation des services sur Redhat

service:

state: started

enabled : yes

name: '{{ item }}'

with_items:

- httpd
- mariadb

```
when: ansible_os_family == 'RedHat'
```

Et les handlers...

Parfois, on souhaite qu'une tâche ne s'exécute que lorsqu'une modification est apportée sur une machine. Par exemple, redémarrer un service si une tâche met à jour la configuration de ce service, mais pas si la configuration est inchangée. C'est le cas des handlers. Et par défaut, les handlers ne s'exécutent qu'une fois que toutes les tâches qui le notifient ont été terminées.

lamp/vars/Debian.yml

```
lamp_packages:
```

- apache2
- mariadb-server
- php
- php-mysql

```
apache_service: apache2
```

lamp/vars/RedHat.yml

```
lamp_packages:
```

- httpd
- mariadb-server
- php
- php-mysqlnd

```
apache_service: httpd
```

lamp/handlers/main.yml

```
---
```

- name: restarting_apache

```
service:
```

```
  name: '{{ apache_service }}'
```

```
  state: restarted
```

lamp/tasks/main.yml

- name: configuration des variables

```
include_vars: "{{ ansible_os_family }}.yaml"

...

- name : activation du mod_rewrite d'apache sur Debian
  community.general.apache2_module:
    state: present
    name: rewrite
  when: ansible_os_family == 'Debian'
  notify:
    - restarting_apache
```

Un dernier pour la route

lamp/vars/Debian.yml

```
lamp_packages:
  - apache2
  - mariadb-server
  - php
  - php-mysql

apache:
  service: apache2
  user: www-data
```

lamp/vars/RedHat.yml

```
lamp_packages:
  - httpd
  - mariadb-server
  - php
  - php-mysqlnd

apache:
```

service: httpd

user: apache

lamp/handlers/main.yml

- name: restarting_apache

service:

name: '{{ apache.service }}'

state: restarted

lamp/tasks/main.yml

- name: configuration des variables

include_vars: "{{ ansible_os_family }}.yml"

...

- name: Mise en place de index.html

template:

src: index.j2

dest: /var/www/html/index.html

owner: "{{ apache.user }}"

group: "{{ apache.user }}"

mode: 0644

lamp/templates/index.j2

```
<!DOCTYPE html>
<html>
<head>
<title>{{ ansible_distribution }} Apache Web Server</title>
<meta charset="utf-8" />
</head>
<body>
<h1>{{ ansible_distribution }} Apache Web Server</h1>
<h2>Site en Constuction</h2>
{% if lamp_packages is defined %}
<h3>Packages :</h3>
<ul>
{% for item in lamp_packages %}
<li>{{item}}</li>
{% endfor %}
</ul>
```

```
{% endif %}
<p>Généré par {{ ansible_distribution }} {{ ansible_distribution_version }}</p>
</body>
</html>
```

Ansible lint

apt install ansible-lint

ansible lint lamp.yml

...

Debian : Ansible et debconf

Il nous faut encore installer **phpmyadmin**...

Sur debian installer debconf-utils

debconf-show phpmyadmin

Avec le playbook correspondant qu'on inclut dans notre tasks/main.yml :

- name: installation de phpmyadmin

include_tasks:

file: '{{ ansible_os_family }}-phpmyadmin.yml'

apply:

tags: phpmyadmin

tags: phpmyadmin

Le mot-clé tags permet d'appeler spécifiquement une tâche sans rejouer tout le playbook :

ansible-playbook playbooks/lamp.yml --tags phpmyadmin

Debian-phpmyadmin.yml

- name: configuration debconf

debconf:

name: phpmyadmin

question: 'phpmyadmin/reconfigure-webserver'

```

    value: 'apache2'
    vtype: select
- name: configuration debconf
  debconf:
    name: phpmyadmin
    question: 'phpmyadmin/mysql/admin-pass'
    value: '(password omitted)'
    vtype: string
- name: configuration debconf
  debconf:
    name: phpmyadmin
    question: 'phpmyadmin/dbconfig-install'
    value : 'true'
    vtype: boolean

- name: installation phpmyadmin
  package:
    name: phpmyadmin
    state: present

```

T.P. Écrire RedHat-phpmyadmin.yml en vous aidant de lampRocky.pdf. Attention, vous allez devoir vous servir du module shell. Veillez à ce que la tâche soit idempotente : ansible-doc shell

Ansible vault

Pour nos tests, nous allons créer une base de données et un utilisateur qui devra pouvoir se connecter à phpmyadmin.

Pour Debian :

```

vi playbooks/test_dbuser.yml
- name: install dépendances python
  package:
    name: python3-pymysql
    state: present

```

```
- name: Create a MySQL DB
community.mysql.mysql_db:
  name: prenom
  state: present
  login_unix_socket: /var/run/mysqld/mysqld.sock
- name: Create a MySQL user
community.mysql.mysql_user:
  name: prenom
  password: secret
  priv: 'prenom.*:ALL'
  state: present
  login_unix_socket: /var/run/mysqld/mysqld.sock
```

Après avoir rapatrié (et modifié en conséquence) notre playbooks dans notre rôle via un import dans tasks/main.yml :

```
- name: configuration des variables
include_vars: '{{ ansible_os_family }}.yaml'
tags :
  - test
...
- name: test dbuser
import_tasks: test_dbuser.yml
tags:
  - never
  - test
```

N.B. Imports (statiques) et includes (dynamiques) :

Toutes les instructions import sont prétraitées au moment de l'analyse des playbook.*

Toutes les instructions include sont traitées au fur et à mesure qu'elles sont rencontrées lors de l'exécution du playbook.*

Le tag never indique qu'il ne doit jamais être exécuté s'il n'est pas appelé spécifiquement avec :

```
ansible-playbook playbooks/lamp.yml --tags test
```

On peut aussi utiliser par exemple :

```
ansible-playbook playbooks/lamp.yml --skip-tags phpmyadmin
```

et ensuite :

```
ansible-playbook playbooks/lamp.yml --tags phpmyadmin, test
```

T.P. Transformer ce playbook pour qu'il prenne en charge RedHat :

Attention au socket et à la librairie pymysql différents sous redhat et debian : il va vous falloir définir des variables...

T.P. faire un playbook clean_dbuser.yml pour nettoyer l'utilisateur et la base de données

Par contre, dans test_dbuser.yml, nous avons un mot de passe en clair !

Encryptage avec un mot de passe

T.P. Modifiez vos playbooks test_dbuser et clean_dbuser pour vous servir de variables que vous renseignerez dans le fichier lamp/defaults/main.yml ainsi :

```
db_user : toto
```

```
db_passwd : secret
```

C'est ce fichier lamp/defaults/main.yml que nous allons maintenant chiffrer !

```
ansible-vault encrypt lamp/defaults/main.yml --vault-id @prompt
```

```
ansible-playbook lamp.yml --vault-id @prompt
```

Encryptage avec une clef de chiffrement dans un fichier

```
echo clef_de_chiffrement > .vault_passwd
```

```
chmod 600 .vault_passwd
```

```
ansible-vault encrypt lamp/defaults/main.yml --vault-password-file .vault_passwd
```

```
ansible-playbook lamp.yml --vault-password-file .vault_passwd
```

N.B. On peut de la même manière le décrypter (decrypt) ou l'éditer (edit)

N.B. On peut aussi chiffrer l'inventaire : par exemple, un inventaire avec des mots de passe en

clair...

```
rocky ansible_host=rocky.local ansible_user=prenom ansible_become=true  
ansible_ssh_pass=secret ansible_become_pass=secret
```

Ansible-galaxy

ansible-galaxy search lamp

ansible-galaxy info rôle

ansible-galaxy install rôle

ansible-galaxy list

ansible-galaxy remove rôle

le rôle est alors installé dans :

.ansible/

N.B. Si vous hébergez vos rôles sur un dépôt **git** privé, vous pourrez l'installer ainsi :

```
ansible-galaxy install git+https://votre_dépôt_git/role.git
```

Exemple :

```
ansible-galaxy install git+https://dev.lan.xlinfo.fr/git/ansible/roles/helloworld.git
```

Pour le tester :

```
ansible-playbook ~/.ansible/roles/helloworld/tests/test.yml
```

ou

```
ansible-playbook ~/.ansible/roles/helloworld/tests/test.yml --check      # n'applique pas les  
changements
```

N.B. On peut aussi, avec ansible-galaxy, installer des collections de modules...

Par exemple :

```
ansible-galaxy collection list
```

```
ansible-galaxy collection install community.general
```

Conclusion

Vous pouvez trouver le dépôt git de ce rôle ici :

git clone <https://dev.lan.xlinfo.fr/git/ansible/ci-cd/ansible-lamp.git/>