



Ansible Vault

Maxence Cuingnet


Ansible Vault

Son objectif : chiffrer les éléments secrets

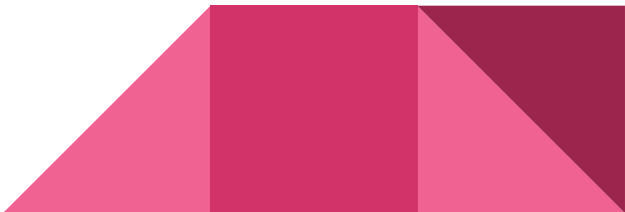
Doc : https://docs.ansible.com/ansible/latest/user_guide/vault.html
<https://docs.ansible.com/ansible/latest/cli/ansible-vault.html>



La commande ansible-vault :

- **create** : crée et ouvre un fichier dans un éditeur, sera chiffré à la fermeture
 - **decrypt** : déchiffre un fichier/variable pour pouvoir l'éditer (ne pas oublier de rechiffrer après avec **encrypt**)
 - **edit** : ouvre un fichier chiffré dans un éditeur (déchiffré), préféré à **decrypt**
 - **view** : voir le contenu d'un fichier chiffré
 - **encrypt** : chiffre un fichier
 - ◆ **encrypt_string** : chiffre une variable (peu recommandé)
 - **rekey** : ré-encrypt avec un nouveau secret
- 

Quelles types de données peuvent être chiffrées ?

- group variables files from inventory
 - host variables files from inventory
 - variables files passed to ansible-playbook with -e @file.yml or -e @file.json
 - variables files loaded by include_vars or vars_files
 - variables files in roles
 - defaults files in roles
 - tasks files
 - handlers files
 - binary files or other arbitrary files
- 

Cas le plus simple : création d'un fichier chiffré


```
mkdir -p group_vars/all
```

```
ansible-vault create group_vars/all/variables.yml
```

```
cat group_vars/all/variables.yml
```

Voir son contenu

```
ansible-vault view group_vars/all/variables.yml
```



Cas le plus simple : création d'un fichier chiffré

Éditer son contenu

```
ansible-vault edit group_vars/all/variables.yml
```

Déchiffrer le fichier

```
ansible-vault decrypt group_vars/all/variables.yml
```

```
cat group_vars/all/variables.yml (ou vim)
```

Ne pas oublier de rechiffrer le fichier :



Cas le plus simple : création d'un fichier chiffré

```
ansible-vault encrypt group_vars/all/variables.yml
```

```
ansible -i "127.0.0.1," all --ask-vault -m debug -a "var=mavARIABLE"
```

Tester également sans l'option `--ask-vault`

ansible ad-hoc commands



Cas le plus simple : utilisation d'une variable chiffrée

INCONVÉNIENT DE CE MODE DE FONCTIONNEMENT : lorsque vous rencontrez des bugs et que vous voulez vérifier les variables ou le contenu de fichier, grep ne peut rien “attraper”

```
cd group_vars  
grep -ri mavariable
```

Il existe dans ce but une meilleure pratique :

```
rm group_vars/all/variables.yml
```

```
sudo vim group_vars/all/variables.yml
```



Cas le plus simple : création d'un fichier chiffré

```
mavARIABLE: "{{ vault_mavARIABLE }}"
```


On ne va pas chiffrer ce fichier, mais nous allons créer un fichier à la racine du précédent contenant cette variable qui sera vaultée

```
ansible-vault create group_vars/all/vault.yml
```

```
vault_mavARIABLE: "ceci est un secret"
```

```
tree
```

Maintenant avec un grep nous pouvons localiser la variable mais on ne peut pas la voir. Le fichier vault est toujours à côté du fichier de la variable



Cas le plus simple : comment éviter de taper le vault_password à chaque fois?

```
ansible -i "127.0.0.1," all --ask-vault -m debug -a "var=mavariable"
```

Cette commande nous contraignait à taper à chaque fois le mot de passe.

Pour éviter cela, on peut appeler un fichier qui contient le vault_password

```
vim ~/.vault_password
```

Et on y écrit en clair le mot de passe. Évidemment, il faut que ce fichier soit bien dissimulé, ne soit pas gité, changer les droits pour éviter que tout le monde puisse le lire

```
chmod 600 ~/.vault_password
```

```
ansible -i "127.0.0.1," all --vault-password-file  
~/.vault_password -m debug -a "var=mavariable"
```

Cas le plus simple : comment éviter de taper le vault_password à chaque fois?

La seconde possibilité est de définir une variable d'environnement afin de ne plus avoir même à demander la consultation du mot de passe dans un fichier

```
export ANSIBLE_VAULT_PASSWORD_FILE=~/.vault_password
```

Pour supprimer cette variable (l'effectuer pour passer à la commande suivante):

```
unset ANSIBLE_VAULT_PASSWORD_FILE
```



QUE VAUT-IL MIEUX ÉVITER DE FAIRE AVEC ANSIBLE_VAULT ?

Il s'agit de la commande **encrypt_string** pour chiffrer une variable

```
ansible-vault encrypt_string 'mon secret' --name 'mysecret'
```

```
sudo vim group_vars/all/variables.yml
```

```
mysecret: !vault |
    $ANSIBLE_VAULT;1.1;AES256
39343466343232646339663264643163386661396237376338383464333235316633613139353032
3437353165613039613031663633313962363436323632370a386637313936313834326334326430
61393565633533636663333662633137646134616365313765616237323761336364616137666537
3263653034303762610a353465396162316535623833393064343065383631363639373132666437
35633862653731333661663433383836353366323235633131623331343033393730
```

QUE VAUT-IL MIEUX ÉVITER DE FAIRE AVEC ANSIBLE_VAULT ?

```
ansible -i "127.0.0.1," all --ask-vault -m debug -a "var=mysecret"
```

Cela fonctionne également bien mais pose des inconvénients. Par exemple lorsque nous voulons modifier cette variable...

```
sudo vim group_vars/all/variables.yml
```

On retombe sur le chiffage de la variable

Seconde possibilité : on peut essayer d'utiliser la commande edit

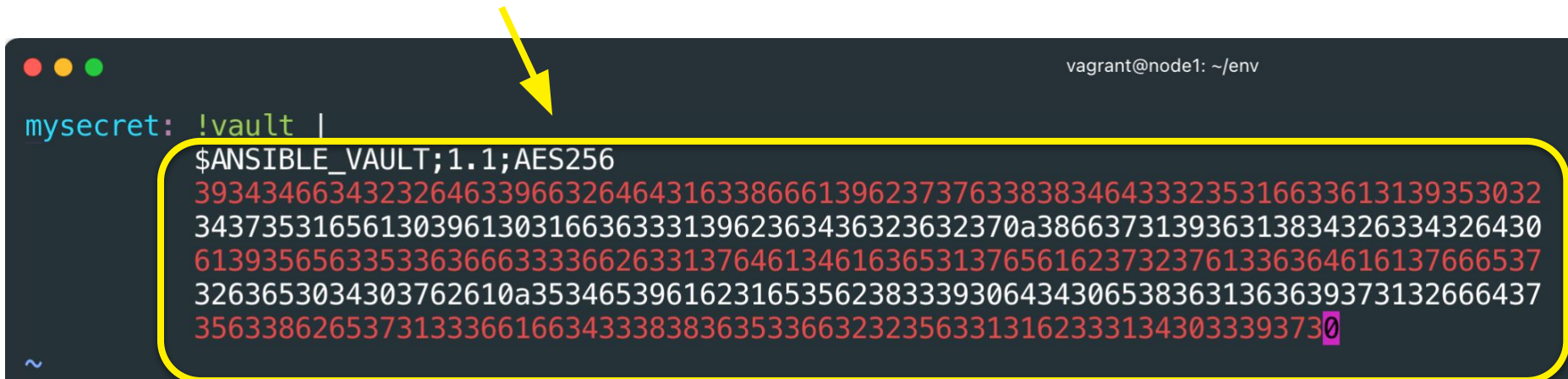
```
ansible-vault edit group_vars/all/variables.yml
```

Mais il nous dit qu'il ne trouve pas de variable chiffrée



QUE VAUT-IL MIEUX ÉVITER DE FAIRE AVEC ANSIBLE_VAULT ?

Pourquoi ? Parce qu'Ansible s'attend juste à trouver cette partie en début de fichier



A terminal window with a dark background. The prompt is 'vagrant@node1: ~/env'. The user has entered 'mysecret: !vault |'. The output shows the decryption of an Ansible Vault file. A yellow arrow points from the text 'Parce qu'Ansible s'attend juste à trouver cette partie en début de fichier' to the start of the vault output. The output is enclosed in a yellow rounded rectangle. The first line is '\$ANSIBLE_VAULT;1.1;AES256'. The following lines are a long string of red characters, which is the decrypted content. The string ends with a pink cursor.

```
vagrant@node1: ~/env
mysecret: !vault |
$ANSIBLE_VAULT;1.1;AES256
39343466343232646339663264643163386661396237376338383464333235316633613139353032
3437353165613039613031663633313962363436323632370a386637313936313834326334326430
61393565633533636663333662633137646134616365313765616237323761336364616137666537
3263653034303762610a353465396162316535623833393064343065383631363639373132666437
35633862653731333661663433383836353366323235633131623331343033393730
```

Même problème avec la commande view

QUE VAUT-IL MIEUX ÉVITER DE FAIRE AVEC ANSIBLE_VAULT ?

L'avantage de cette commande **encrypt_string** est que le grep peut tout de même retrouver la variable

```
grep -ri mysecret
```

Donc finalement il est préférable de faire comme précédemment :

```
rm group_vars/all/variables.yml
```

```
sudo vim group_vars/all/mavARIABLE.yml
```

```
mavARIABLE: "{{ vault_mavARIABLE }}"
```



QUE VAUT-IL MIEUX ÉVITER DE FAIRE AVEC ANSIBLE_VAULT ?

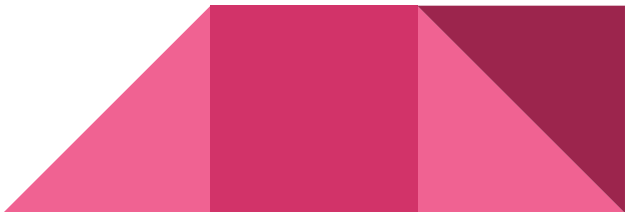
Nous allons maintenant mettre la valeur de notre variable

```
sudo vim group_vars/all/vault.yml
```

```
vault_mavariabe: "voici mon secret"
```

Il nous reste maintenant à chiffrer le fichier

```
ansible-vault encrypt group_vars/all/vault.yml
```



QUE VAUT-IL MIEUX ÉVITER DE FAIRE AVEC ANSIBLE_VAULT ?

```
cat group_vars/all/mavARIABLE.yml
```

```
grep -ri mavARIABLE
```



CLI VAULT : Vault ID et MULTI-PASSWORD

Vault-id est un moyen d'identifier des passwords

```
sudo vim group_vars/all/data.yml
```

```
mysecret: "ceci est mon secret"
```

Puis on va éditer un fichier contenant notre password. Bien sûr, il faut normalement le faire dans une zone plus retirée

```
vim pwd.txt
```

→ y écrire le password



CLI VAULT : Vault ID et MULTI-PASSWORD

Nous allons donc utiliser ce password pour chiffrer notre fichier data

```
ansible-vault encrypt --vault-id pwd.txt group_vars/all/data.yml
```

```
ansible-vault edit --vault-id pwd.txt group_vars/all/data.yml
```

Autre option possible :

```
ansible-vault edit --vault-id @prompt group_vars/all/data.yml
```



CLI VAULT : Vault ID et MULTI-PASSWORD


On peut également l'utiliser dans la ligne de commande :

```
ansible -i "127.0.0.1," all --vault-id pwd.txt -m debug -a "msg='{{mysecret}}'"
```

Maintenant, nous allons pouvoir utiliser des ID

```
mv pwd.txt pwd-data.txt
```

```
sudo vim group_vars/all/data.yml
```



CLI VAULT : Vault ID et MULTI-PASSWORD

Ajout de l'ID, ici **data**



```
$ANSIBLE_VAULT;1.1;AES256;data
```

```
36336163303963643633336630653130623063346433643963663461386662323138313763373330  
6463306432303765666566653038643262376461313365380a313637653132623263306430633937  
61363032306634613563643862306462613166626134643562383334636562306362353163303634  
6538616364303332630a323835383766373934633430366631383335656462393237383334303632  
63646138613830326136366361333939386462336232376465646435643961323962346266313066  
3037623038646661346333353265393032323566376635336530
```

```
~
```

```
~
```

vagrant@node1: ~/env

CLI VAULT : Vault ID et MULTI-PASSWORD

On va pouvoir ensuite préciser l'ID dans notre CLI

```
ansible -i "127.0.0.1," all --vault-id data@pwd-data.txt -m debug -a "msg='{{mysecret}}'"
```

L'intérêt apparaît lorsque nous avons plusieurs mots de passes

```
- name: test
  hosts: all
  connection: local
  tasks:
    - name: debug
      debug:
        msg: "{{montitre}} {{mysecret}}"
```

playbook.yml avec deux mots
de passes
ID mysecret : data
ID montitre : app



CLI VAULT : Vault ID et MULTI-PASSWORD

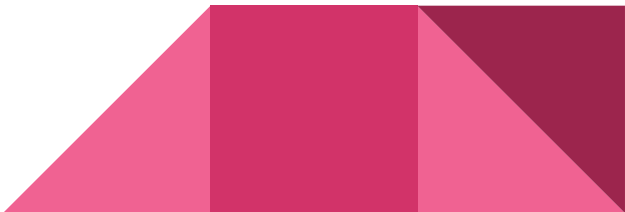
```
sudo vim group_vars/all/app.yml
```

```
montitre: "Titre1 :"
```

Et maintenant, nous allons chiffrer nos éléments, pour cela nous créons, avec la même méthode, un second fichier de password

```
sudo vim pwd-app.txt
```

→ y écrire le password



CLI VAULT : Vault ID et MULTI-PASSWORD


Nous allons maintenant chiffrer nos deux fichiers contenant les passwords

```
ansible-vault encrypt --vault-id app@pwd-app.txt group_vars/all/app.yml
```

Nous vérifions le chiffrement, notez que l'ID a bien été rajoutée

```
sudo vim group_vars/all/app.yml
```

Nous avons donc maintenant deux passwords différents pour chacune des variables



CLI VAULT : Vault ID et MULTI-PASSWORD

```
$ANSIBLE_VAULT;1.2;AES256;app  
61373237313461343137313561333766623  
34326330336136346433383939333564323  
39313838393437623661626235353032316  
3931313133626235350a343761353730633  
38623238353636383137383132633735303
```

```
$ANSIBLE_VAULT;1.1;AES256;data  
36336163303963643633336630653130623063346  
64633064323037656665666530386432623764613  
61363032306634613563643862306462613166626  
6538616364303332630a323835383766373934633  
63646138613830326136366361333939386462336  
30376230386466613463333532653930323235663
```

CLI VAULT : Vault ID et MULTI-PASSWORD

Maintenant que nous avons deux mots de passes, nous avons besoin de vault-id pour les passer en CLI avec ansible-playbook

```
ansible-playbook -i "127.0.0.1," --vault-id data@pwd-data.txt --vault-id app@pwd-app.txt playbook.yml
```

```
TASK [Gathering Facts] *****
ok: [127.0.0.1]

TASK [debug] *****
ok: [127.0.0.1] => {
  "msg": "Titre1 : ceci est mon secret"
}

PLAY RECAP *****
127.0.0.1 : ok=2    changed=0
```